DREAM.3D
User Manual and Filter Reference

DREAM3D Development Team

dream3d@bluequartz.net

June 10, 2013

Contents

1 Table of Contents **3**

2 Brief History and Acknowledgements **5**

3 DREAM.3D Data Structure **9**

4 Overview of the User Interface **13**

5 Creating a Pipeline **17**

6 Import, Export & Favorite Pipelines **21**

7 Supported File Formats **23**

7.1 Native DREAM3D File Format . . . 23

7.2 H5EBSD Data File Specification . . . 25

7.3 XDMF For Visualization . . . 30

7.4 Import & Export File Formats . . . 32

8 Tutorials **33**

8.1 3D Orientation Data Reconstruction . . . 33

8.2 Synthetic Microstructure Generation . . . 35

8.3 Surface Meshing Tutorial . . . 38

8.4 Cropping Data Tutorial . . . 42

9 Auxiliary Tools **45**

9.1 Stats Generator . . . 45

10 Created Array Index **55**

CHAPTER 1

Table of Contents

The DREAM3D Rererence Manual contains all the information to start using DREAM3D quickly. The user is urged to read through the short introduction to get a basic idea of how to use DREAM3D. For a really quick start the user is referred to the various tutorials that have been created. For those uses importing data from other sources the sections on the various file formats will be of interest. Finally the "Filter Documentation" lists each of the filters that the user can utilize in their workflows.

Welcome to DREAM.3D, an open and modular software package that allows users to reconstruct, instantiate, quantify, mesh, handle and visualize microstructure digitally. The core software package has four (4) major functionalities.

- First, a series of EBSD data files (either TSL .ang or HKL .ctf) or images (.bmp, .jpeg, .tif or .png) can be read and packed into a single HDF5 data file in order to reduce data size and clutter.
- The sections can be aligned, cleaned, reconstructed, segmented and statistically characterized.
- Statistics, either synthetically created or from real data, can be used to generate a statistically equivelent microstructure.
- The reconstructed volumes and the synthetic volumes can be surfaced meshed to allow the models to be moved into FEM or other simulations.

DREAM3D makes the reconstruction of 3D EBSD data simple and straight forward. The development of more features is always on going and the DREAM3D development team welcomes your feed back whether you are first time user or seasoned user.

Brief History and Acknowledgements

DREAM.3D Data Structure

Supported File Formats

Overview of the User Interface

Creating a Pipeline

Import, Export & Favorite Pipelines

filterdocumentation

Created Array Index

Tutorials

Auxiliary Tools

CHAPTER 2

Brief History and Acknowledgements

History of DREAM.3D

The history of DREAM.3D is rooted in two parallel research efforts:

- The first was the graduate research project of Michael Groeber, under the advisement of Dr. Somnath Ghosh, at The Ohio State University (2003-2007). The key developments under this project were the creation of many of DREAM.3D's current filters for reconstructing serial-section experimental datasets, quantifying microstructural features in 3D and generating statistically equivalent microstructures.

- The second research effort was carried out by the research group of Tony Rollett at Carnegie Mellon University over a number of graduate student projects (see names below) and years ($\sim$2000-2011). The key developments under these projects were the development of a synthetic microstructure builder, microstructure clean-up tools, surface mesh generation and quantification tools for microstructural features.

The tools developed under these efforts generally remained as command-line executables with limited portability due to lacking documentation. In an effort to extend the capabilites of these tools to the larger materials community, Dr. Chris Woodward of the Air Force Research Laboratory (AFRL) funded Mike Jackson of BlueQuartz Software to create Graphical User Interfaces (GUIs), common data structures and documentation for the tools developed in the research effort of Groeber and Ghosh (2009-2011). After the initial phase of development (2011), Jackson and Groeber approached Rollett, Lee, Sintay, Chan and Tucker and proposed the unification of the parallel efforts. The Rollett group accepted the proposal and graciously made available all of their algorithms and source code. It was at this point that DREAM.3D was named and the vision for the software was truly solidified.

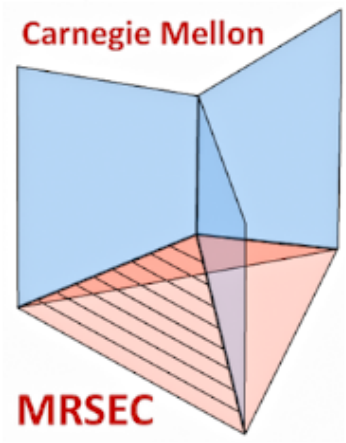
Over the next couple years (2011-2013), additional researchers contributed code for extending capabilites of DREAM.3D. Specifically, the grain boundary-centric tools of Dave Rowenhorst (Naval Research Laboratory) and Greg Rohrer (Carnegie Mellon University) greatly augmented the functionality of the package.

By 2013, Jackson and Groeber had fully documented the tools, created a Software Developer's Kit (SDK) and generated tools for adding plugins to the package - making the software package readily usable and extensible.

Funding Sources

Parts of DREAM.3D were written under US Air Force (AFRL) Contract FA8650-07-D-5800 and Navy Research Labs (NRL) Contract N00173-07-C-2068. Many others have contributed to the code including the following:

Major Contributing Institutions



List of Code Contributors (CC) and Vision Contributors (VC)

Air Force Research Laboratory (AFRL)

- Dr. Michael A. Groeber (VC, CC)
- Dr. Adam Pilchak (VC, CC)
- Dr. Megna Shah (VC, CC)
- Dr. Michael Uchic (VC)
- Dr. Chris Woodward (VC)
- Dr. Dennis Dimiduk (VC)

BlueQuartz Software

- Mr. Michael A. Jackson (VC, CC)

Ohio State University/Johns Hopkins University

- Dr. Somnath Ghosh (VC)

Carnegie Mellon University

- Dr. Tony Rollett (VC, CC)
- Dr. Greg Rohrer (CC)
- Dr. Marc De Graef (CC)
- Dr. Sukbin Lee (CC)
- Dr. Joe Tucker (CC)
- Dr. Lisa Chan (CC)
- Dr. Stephen Sintay (CC)
- Dr. Abhijeet Brahme (CC)
- Dr. David Saylor (CC)
- Dr. Joe Fridy (CC)
- Dr. Patrick Callahan (CC)
- Dr. Shlomo Taasan (CC)

Navy Research Laboratory (NRL)

- Dr. Dave Rowenhorst (VC, CC)

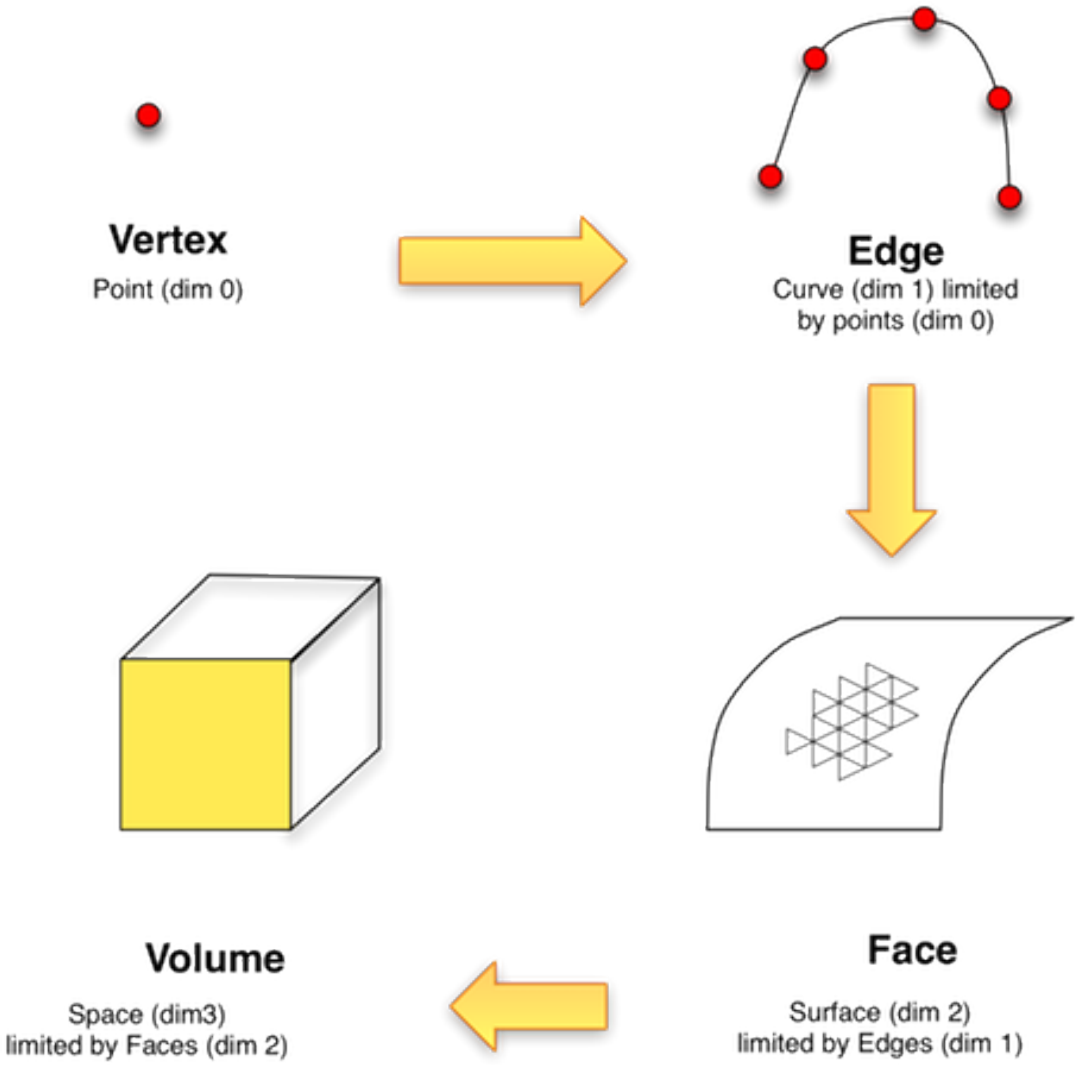
Other Insitituations

- Mr. Joey Kleingers (CC) (SOCHE)
- Mr. Jarrell Waggoner (CC) (Univ. South Carolina)

CHAPTER 3

DREAM.3D Data Structure

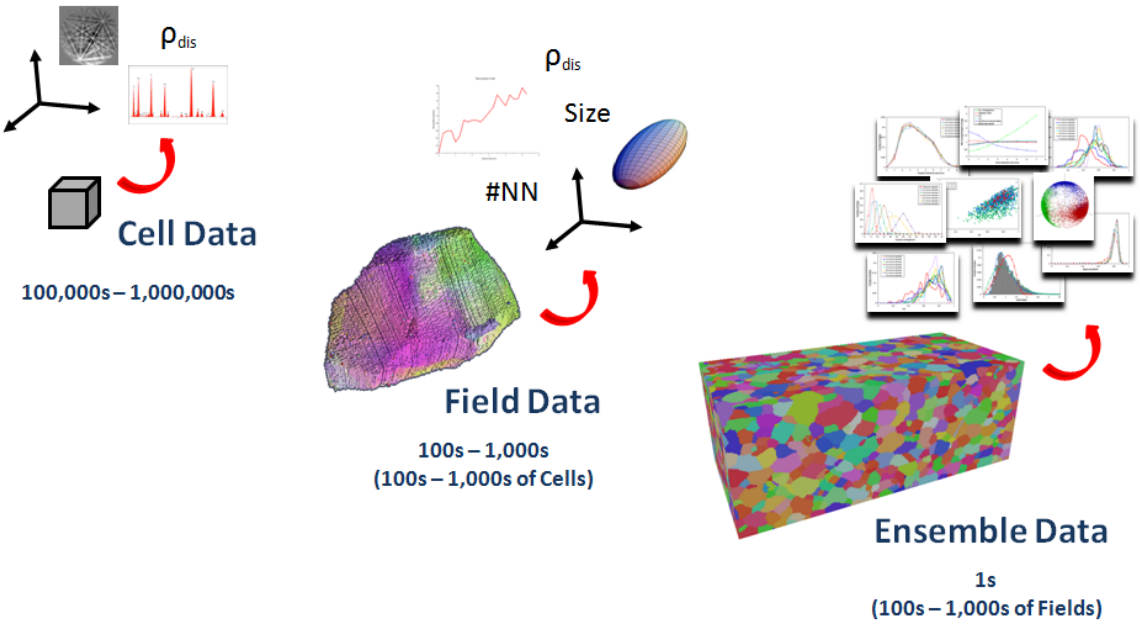
DREAM3D uses a data structure that is based on the concepts of combinatorial topology and common methods for describing mesh structures. Any topological network (in 2- or 3-D) can be decribed with a the following hierarchy:



If the mesh (or network) is 2-D, then the *Volume* element does not exist and the *Face* element is the highest level element. For typical *voxel-based* data, the *voxels* are the *Volume* elements and are referred to within DREAM.3D as **Cells**. Similarly, in a surface mesh with triangular patches, the triangles are the *Face* elements and are referred

to within DREAM.3D as **Faces**. This topology is required to describe the "mesh" or "structure" of the data itself and is not/cannot be adjusted/defined by the user.

However, once the topology of the "structure" is set, the user can begin *grouping* topological elements into higher level features of the "structure". Below is an example of this grouping for a *voxel-based* dataset of a polycrystalline metal.



DREAM.3D uses two additional levels above the topological levels required to define the "structure" of the data. These two levels are called: **Field** and **Ensemble**. These levels allow the user to *group* topological elements together based on criteria of similarity. For the example above, the **Cells** are *grouped* to identify **Fields** (i.e. grains) by applying a criterion of similar crystallographic orientation (i.e. neighboring **Cells** with similar orientation are said to belong to the same **Field**). The **Fields** are then *grouped* to identify **Ensembles** by a criterion of similar phase (i.e. all grains of the same phase in the dataset are said to belong to the same **Ensemble**). The grouping criteria are at the discretion of the user, because these additional levels are not required for description of the "structure", but rather are organizational levels for describing the information that lives on the "structure".

At each level, DREAM.3D creates a *Map* to store information/data about the individual elements at that level. Additional information about the maps of the DREAM.3D data structure and "typical" data are given below:

Vertex

- Map of attributes associated with single *points*.
- Measured values (i.e. Orientation, Chemistry, Greyscale, etc.) - generally measured data is not represented at the **Vertex** level, but the user could choose to store measured values at the **Vertices** rather than the more typical **Cell**-centered convention.
- Calculated values (i.e. Curvatures, Coordination, etc.) - these calculated values are typically values associated with the connectivity or geometry of the "mesh" or "structure" itself.

Edge

- Map of attributes associated with edges of **Faces**.
- Measured values - data is not typically measured at the **Edge** level.
- Calculated values (i.e. Curvatures, Coordination, etc.) - these calculated values are typically values associated with the connectivity or geometry of the "mesh" or "structure" itself.

Face

- Map of attributes associated with surface patches.
- Measured values - data is not typically measured at the **Face** level.
- Calculated values (i.e. Bounding Field Ids, Normals, Curvature, etc.) - these calculated values are typically IDs to the higher level maps (**Field** and **Ensemble**), related to gradients of the measured values, values associated with the connectivity or geometry of the "mesh" or "structure" itself or relationships to values calculated in the higher level maps.

Cell

- Map of attributes associated with single *datapoints* - often these *datapoints* are not truly *volumes*, but rather point-probe measurements that are homogenized over the volume nearest to each probe-point.
- Measured values (i.e. Orientation, Chemistry, Greyscale, etc.) - typically this is the level at which most data is actually acquired.
- Calculated values (i.e. Field Ids, Kernel Avg. Misorientation, Euclidean Distance, etc.) - these calculated values are typically IDs to the higher level maps (**Field** and **Ensemble**), related to gradients of the measured values or relationships to values calculated in the higher level maps.

Field

- Map of attributes associated with sets of *datapoints* - generally *elements* in this map are sets of contiguous **Cells** that have some aspect of similarity, often in their measured values (i.e. orientation, chemistry, greyscale, etc.).
- Measured values (i.e. Avg. Stresses, Avg. Strains, etc) - some measurement techniques are capable of measuring values averaged over an entire **Field** (i.e. High Energy Diffraction Microscopy).
- Calculated values (i.e. Avg. Orientation, Size, Shape, No. of Neighbors, etc.) - generally there is little measured data at this level and these values are calculated from averaging the measured data from the **Cells** that constitute each **Field** or geometrically describing the size and/or shape of the set of **Cells**.

Ensemble

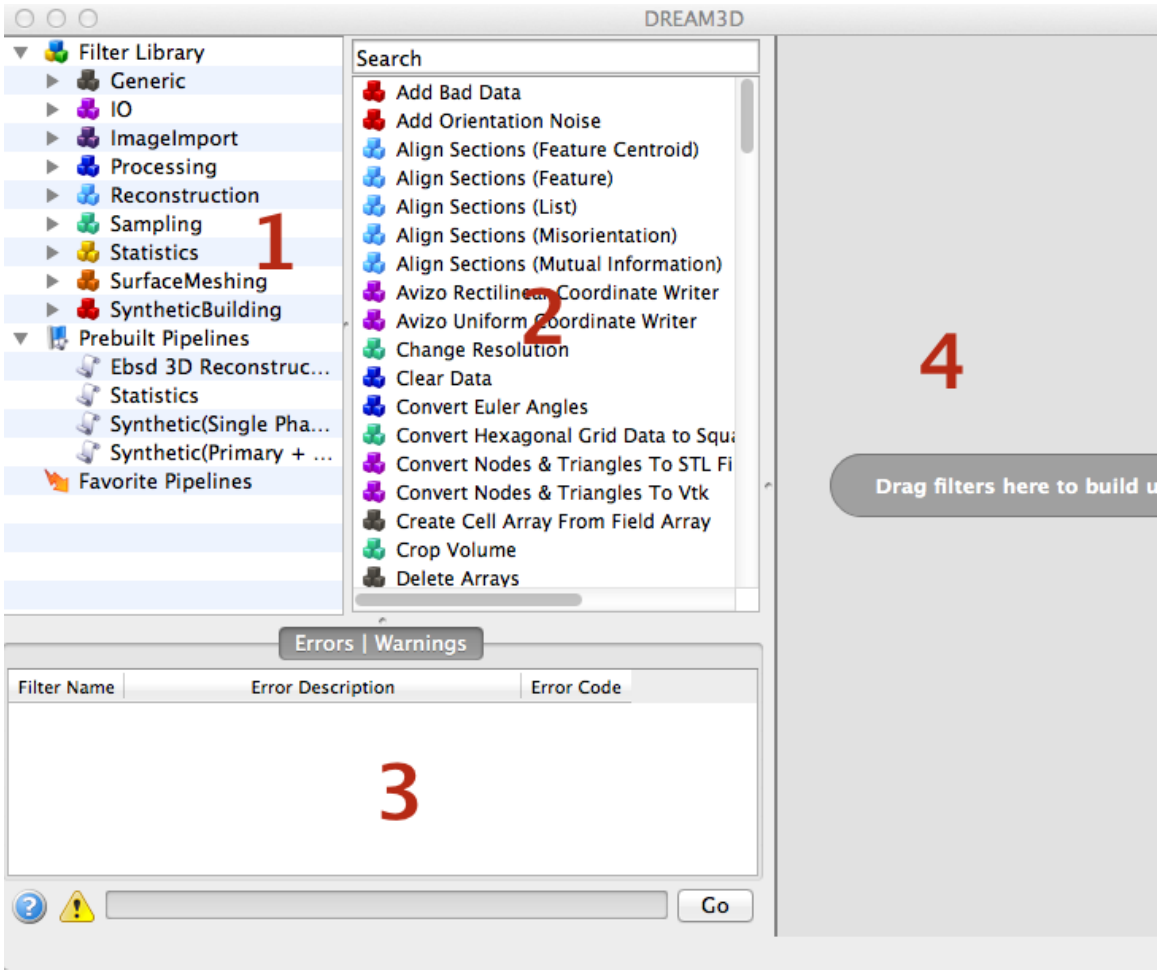
- Map of attributes associated with sets of **Fields**
- Measured/Set values (i.e. Crystal Structure, Phase Type, etc.) - measured values at the **Ensemble** level are usually user-defined parameters during the data collection or during analysis within DREAM.3D.
- Calculated values (i.e. Size Distribution, No. of Fields, ODF, etc.) - these values are typically descriptions of the distribution of values from the lower level **Field** map.

CHAPTER 4

Overview of the User Interface

DREAM3D has 4 main areas of its user interface:

1. The Filter Library
2. The Filter List
3. The Errors Tab
4. The Pipeline Area



The Filter Library

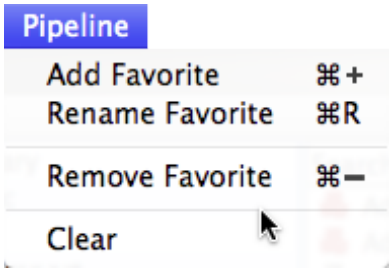
Area 1: This area of the user interface is broken into 3 basic groups: *Filter Library*, *Prebuilt Pipelines* and *Favorite Pipelines*. The *Filter Library* section contains the various groups that the filters are organized into. The user can click a specific group and just the filters associated with that group will be displayed in the *Filter List* area of the user interface.

Prebuilt Pipelines

DREAM3D provides several *Prebuilt Pipelines* that can aid the new user in getting started with DREAM3D. Simply double clicking a preset will clear any current filters in the pipeline area and populate the pipeline area with the filters from the Pipeline Preset.

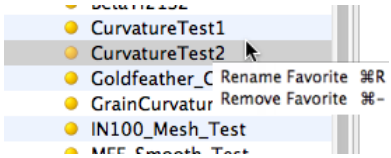
Favorite Pipelines

After the user builds a pipeline that they may want to save for later the user can use the *Pipeline* menu and select the "Add Favorite" menu to save that specific pipeline configuration. If at a future point in time the user wants to remove the favorite from the *Favorites* list they user can select from the "Add Favorite" menu and choose *Remove Favorite* to remove it from the list.



Additional shortcuts are available by *right clicking* on the *Favorite Pipelines* tree item which will display a contextual menu to create a new favorite pipeline entry.

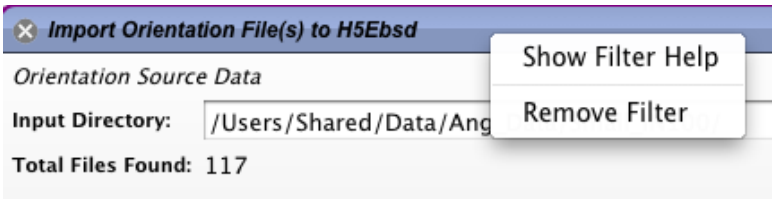
The user can also rename a *Favorite Pipeline* entry by right clicking on the entry item and selecting the desired action. The only valid characters that can be used are AlphaNumeric, the underscore (_) and the dash (-).



The user should also note that simply selecting a *Favorite Pipeline* from the list will display the filters in order of operation in the *Filter List* area.

Getting Help for a Filter

The user can *Right Click* in the title area of the filter to display a menu where the user can select the "Show Filter Help" menu selection. The user's default web browser will be opened to the page containing the documentation for the current filter.

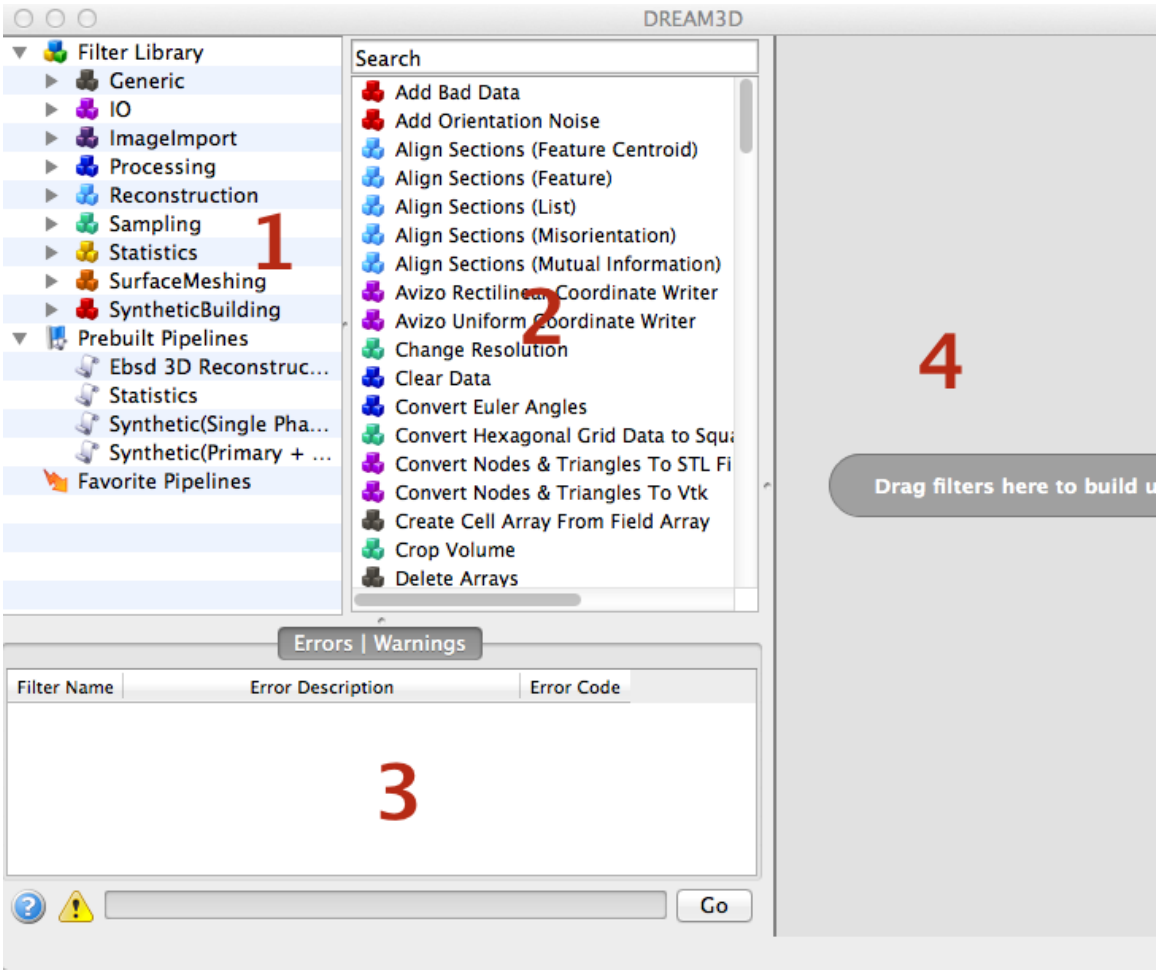


Filter List

Area 2: This section lists the filters that are associated with a specific group that is selected in the *Filter Library* area. If the *Filter Library* is itself selected then all filters will be shown in this list. The user typically will select a filter from this list and drag it over to the pipeline area.

Errors & Warnings Tab

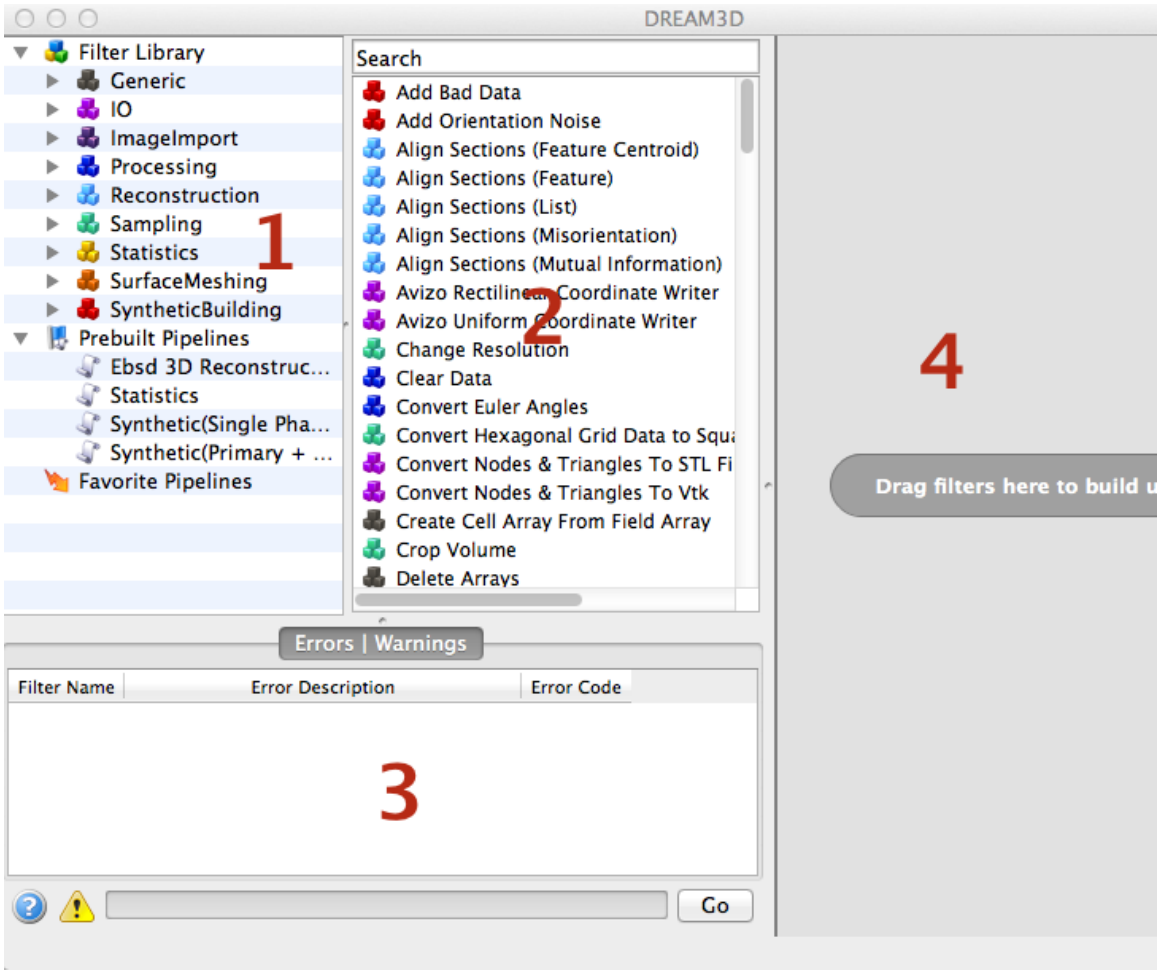
Area 3: This area displays any errors and/or warnings associated with the filter if it is actively being used in a pipeline. The display of the errors can be toggled on and off by pressing the appropriate buttons. Also clicking the ∗∗?∗∗ icon will display all of the help (including filter reference, tutorials and user manual) in the users default web browser.



Pipeline Area

Area 4:This area is where the user will construct their pipeline by either double clicking on a filter in the filter list area or dragging a filter from the filter list and dropping the filter into the pipeline area. Filters in this section can be

rearranged by simply dragging the filter into a new location. As the user updates the pipeline a process called the **Preflight** will be executed to make sure that the pipeline will have all the necessary data available during the actual execution of the processing pipeline.



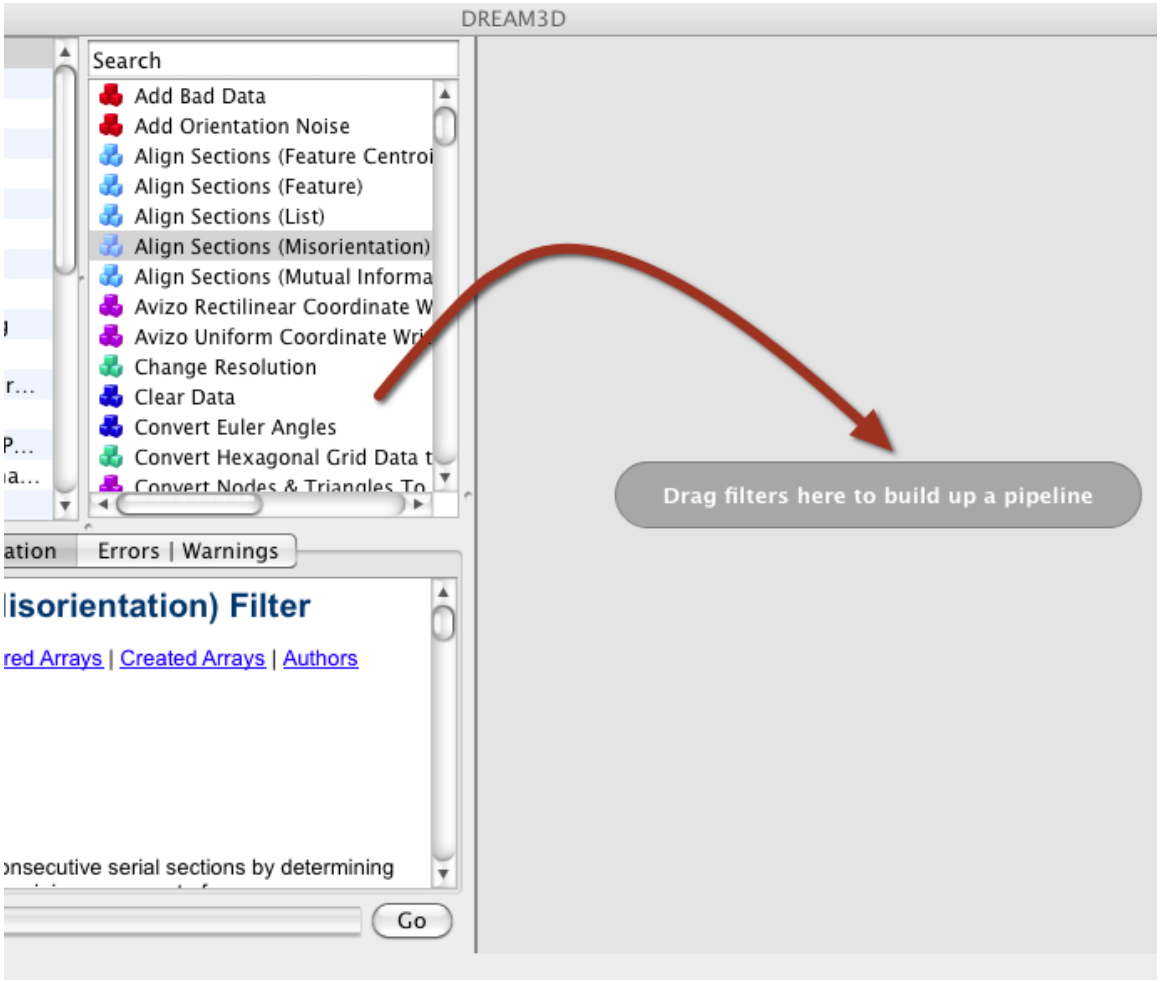
CHAPTER 5

Creating a Pipeline

In DREAM3D the user processes their data by creating what is known as a *Pipeline* which is constructed out of a series of *Filters*. By chaining together a series of filters the data is processed in quick succession and can ultimately be saved to a number of different file formats. The user should be aware that all processing on the data is done **in place**, i.e., there is only a single copy of the data resident in memory at any one time.

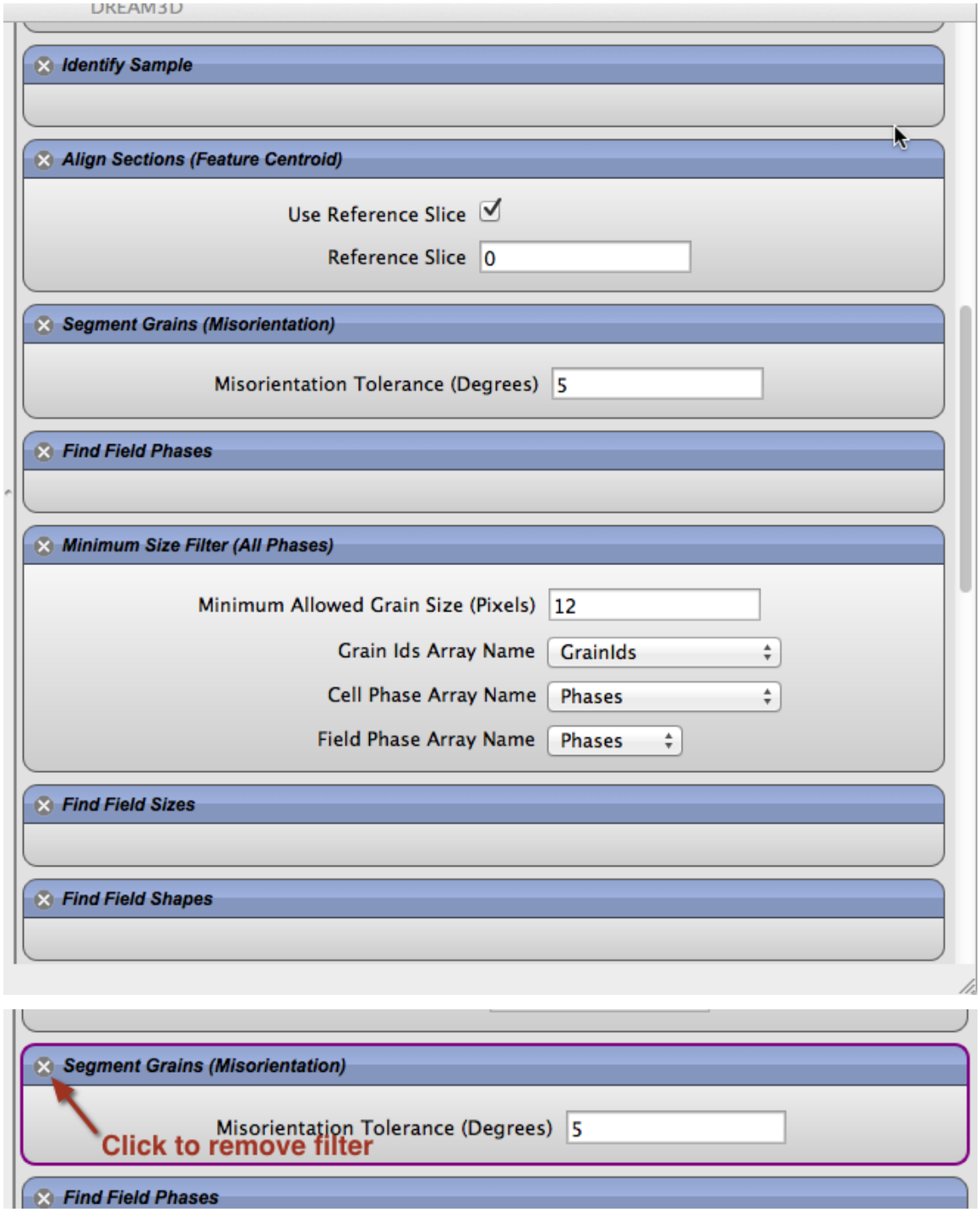
Building a Pipeline

In order the build a pipeline the user can either double click on a particular filter or drag the filter from the *Filter List* into the *Pipeline Area*.



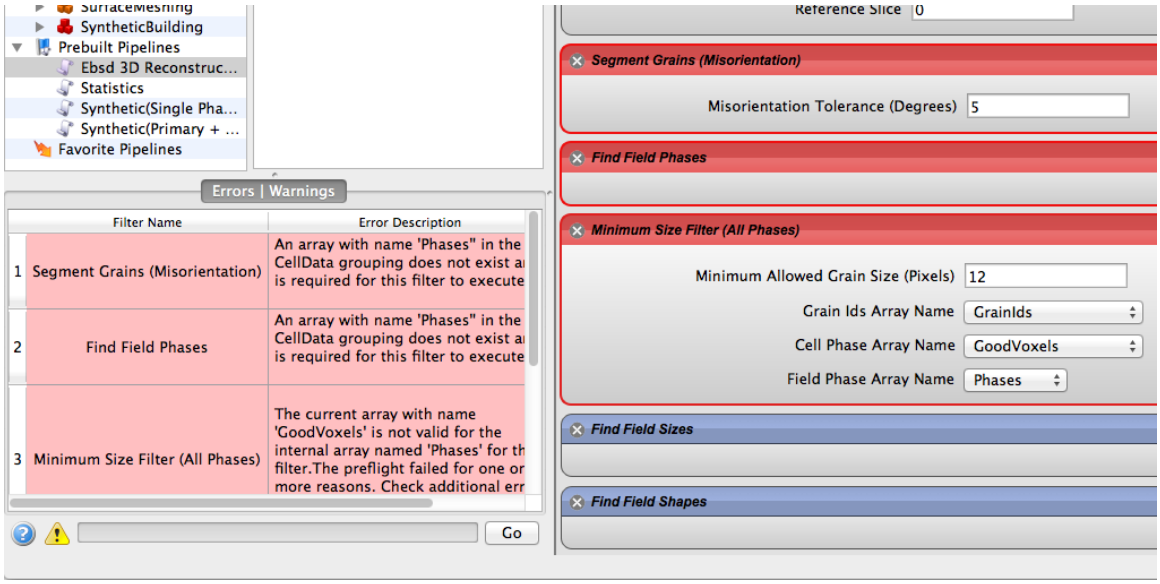
The user can place multiple filters into the pipeline to generate a workflow of data input, analysis and data output

filters. The user can reorder the filters by simply using drag-and-drop of a selected filter in the pipeline and moving it into a new position in the pipeline. As filters are placed into the pipeline a basic error checking algorithm called the **Preflight** is executed on the entire pipeline. During the preflight stage the pipeline will have each filter check its input parameters and the availability of required data arrays. Figure 2 shows a pipeline ready to execute with no errors.



To remove a filter from the pipeline the user can simply use the mouse to click the small **X** icon in the upper left corner of the filter.

If the preflight did not complete successfully the user will see the offending filters show a red outline and red title area. This indicates that one or more of the input parameters have an error or the filter requires data that will not be available at that point in the pipeline. By looking at the error table the user can read the error message from the pipeline and act accordingly. An example pipeline with errors is shown in figure 4.


SyntheticBuilding
Prebuilt Pipelines
Ebsd 3D Reconstruc...
Statistics
Synthetic(Single Pha...
Synthetic(Primary + ...
Favorite Pipelines
Errors | Warnings
Filter Name
Error Description
1 Segment Grains (Misorientation)
An array with name 'Phases" in the CellData grouping does not exist a is required for this filter to execute
2 Find Field Phases
An array with name 'Phases" in the CellData grouping does not exist a is required for this filter to execute
3 Minimum Size Filter (All Phases)
The current array with name 'GoodVoxels' is not valid for the internal array named 'Phases' for th filter.The preflight failed for one or more reasons. Check additional err
Go
Reference Slice 0
Segment Grains (Misorientation)
Misorientation Tolerance (Degrees) 5
Find Field Phases
Minimum Size Filter (All Phases)
Minimum Allowed Grain Size (Pixels) 12
Grain Ids Array Name GrainIds
Cell Phase Array Name GoodVoxels
Field Phase Array Name Phases
Find Field Sizes
Find Field Shapes

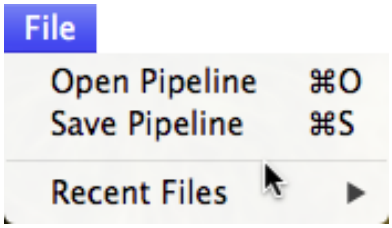
Once the errors are resolved the user can now execute the pipeline and generate the outputs. Because the DREAM3D native file format stores the complete state of the data arrays the user can use this idea to add *checkpoints* to the pipeline in the case of a long running pipeline.

CHAPTER 6

Import, Export & Favorite Pipelines

Saving and Loading a Predefined Pipeline

DREAM3D allows the user to *Export* and *Import* pipelines via the **File** menu and the **Save Pipeline** and **Open Pipeline** menu items. The pipeline files are saved in the _.ini_ file format popular among many other programs. This file format is a simple text file with delineated sections with *Key::Value* entries. This makes the sharing of Pipelines mostly transparent between users. If a pipeline has filters that require files to be read from or written to and the pipeline file is passed to another user that does not have those same paths then the new user may have to make the adjustments inside of DREAM3D after the pipeline is loaded.



The following is a partial listing of a 3D EBSD reconstruction pipeline saved as a pipeline file:

```
[PipelineBuilder]
Number_Filters=15
Name=Reconstruction

[0]
Filter_Name=ReadH5Ebsd
InputFile=/Users/Shared/Data/Ang_Data/Small_IN100_Output/Small_IN100.h5ebsd
ZStartIndex=1
ZEndIndex=117
UseTransformations=false
ArraySelections_VoxelCell\size=4
ArraySelections_VoxelCell\1\VoxelCell=Confidence Index
ArraySelections_VoxelCell\2\VoxelCell=EulerAngles
ArraySelections_VoxelCell\3\VoxelCell=Image Quality
ArraySelections_VoxelCell\4\VoxelCell=Phases
ArraySelections_VoxelField\size=0
ArraySelections_VoxelEnsemble\size=2
ArraySelections_VoxelEnsemble\1\VoxelEnsemble=CrystalStructures
ArraySelections_VoxelEnsemble\2\VoxelEnsemble=MaterialName
ArraySelections_SurfaceMeshPoint\size=0
ArraySelections_SurfaceMeshFace\size=0
ArraySelections_SurfaceMeshEdge\size=0
ArraySelections_SolidMeshPoint\size=0
ArraySelections_SolidMeshFace\size=0
ArraySelections_SolidMeshEnsemble\size=0

[1]
Filter_Name=MultiThresholdCells
OutputArrayName=GoodVoxels
ComparisonInputs\size=2
ComparisonInputs\1\ArrayName=Image Quality
```

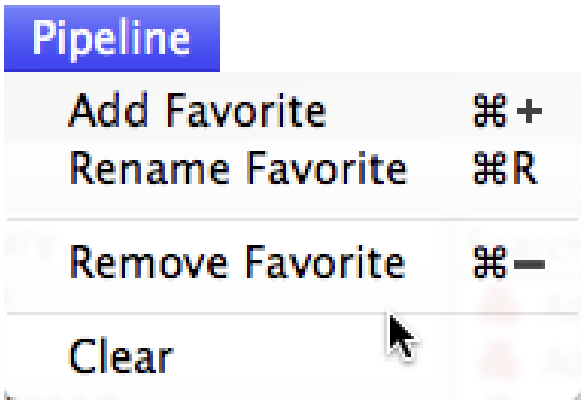
```
ComparisonInputs\1\CompOperator=1
ComparisonInputs\1\CompValue=120
ComparisonInputs\2\ArrayName=Confidence Index
ComparisonInputs\2\CompOperator=1
ComparisonInputs\2\CompValue=0.100000001490116

[2]
Filter_Name=IdentifySample

[3]
Filter_Name=AlignSectionsFeatureCentroid
UseReferenceSlice=true
ReferenceSlice=0
```

Creating a Favorite Pipeline

If the user creates a pipeline that they intend to use several times then they may want to save the pipeline to the *Favorites* area. This can simply be accomplished by using the **Pipeline** menu and selecting the **Add Favorite**. If the user wants to remove a favorite at any time they simply need to select the **Remove Favorite** menu item.



Clearing the Pipeline

As a convenience for those situations where the user has created a large pipeline and they would like to delete every filter from the pipeline the **Pipeline Menu** also has a menu item to remove every filter from a pipeline and an associated keyboard short cut.

CHAPTER 7

Supported File Formats

DREAM3D supports the reading from and writing to a number of different file formats from data acquisition systems to visualization systems. The various file formats are discussed in this section.

Native DREAM3D File Format

A discussion of the native DREAM3D file format based on HDF5.

H5EBSD Data File Specification

A discussion of the H5EBSD file format

XDMF For Visualization

How to use the Xdmf wrapper for visualizing data with ParaView

Import & Export File Formats

This sections lists the various file formats that can be used as import and export of data.

7.1 Native DREAM3D File Format

HDF5

DREAM3D uses the HDF5 as its native file format which means that **_ANY_** software package capable of reading an HDF5 file can read a DREAM3D file. HDF5 stores data in a hierarchical format with complete descriptors for the data stored in the file. HDF5 is open-source and a number of tools exist that allow a user to view and manipulate HDF5 data files. Once such tool is the free HDFView from "The HDF Group" which is a java based program that can view and edit HDF5 data files.

Each array is stored as an individual data set in HDF5 under one of several different types of *DataContainer* storage groups.

- Voxel DataContainer
- SurfaceMesh DataContainer
- SolidMesh DataContainer

Depending on the type of data created the actual data array that a user may be interested in may be stored in various subgroups in the HDF5 file. For example with a Voxel Data Container there are six (6) types of Data: Vertex, Edge, Face, Cell, Field and Ensemble. Arrays created for each type are stored in subgroups. For example with Voxel based data the following HDF5 group organization is used:

- VoxelDataContainer (*Group*)
  - DIMENSIONS (*Dataset*)

  - ORIGIN (*Dataset*)
  - SPACING (*Dataset*)
  - VERTEX_DATA (*Group*)
  - EDGE_DATA (*Group*)
  - FACE_DATA (*Group*)
  - CELL_DATA (*Group*)
    - GrainIds (*Dataset*)
    - EulerAngles (*Dataset*)
    - IPFColors (*Dataset*)
  - FIELD_DATA (*Group*)
    - Phases (*Dataset*)
    - AvgEulerAngle (*Dataset*)
  - ENSEMBLE_DATA (*Group*)
    - CrystalStructures (*Dataset*)
    - MaterialName (*Dataset*)

In the above example we have shown a number of data arrays for each group type. The user should note that the data sets shown in the above example are hypothetical and may or may not appear in data sets that the user generates.

With a SurfaceMesh Data Container there are only five (5) types of Data: Vertex, Edge, Face, Field and Ensemble. The missing type of Data in the SurfaceMesh is the Cell. This is because the SurfaceMesh is a 2-D feature and thus the Face is the Cell and thus a level can be ommitted.

- SurfaceMeshDataContainer (*Group*)
  - Faces (*Dataset*)
  - Vertices (*Dataset*)
  - VERTEX_DATA (*Group*)
    - VertexNormal (*Dataset*)
  - EDGE_DATA (*Group*)
  - FACE_DATA (*Group*)
    - Normals (*Dataset*)
    - Centroids (*Dataset*)
    - Curvature (*Dataset*)
    - IPFColor (*Dataset*)
  - FIELD_DATA (*Group*)
    - AvgCurvature (*Dataset*)
    - MeanWidth (*Dataset*)
  - ENSEMBLE_DATA (*Group*)
    - GBCD (*Dataset*)

In the above we show how a surface mesh is stored. For more information on the actual data structure of the surface mesh please see the Surface Meshing Tutorial.

NOTE: DREAM3D is perfectly capable of storing **BOTH** the voxel data and the surface mesh data in the same file if the user so desires. This allows all the data to stay together and more easily shared among collaborators.

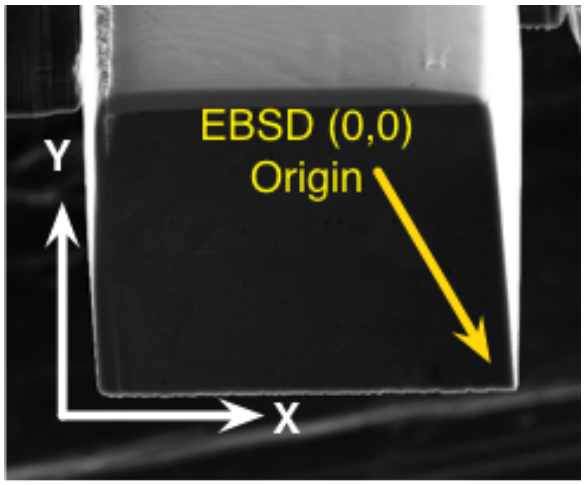
7.2 H5EBSD Data File Specification

Introduction

The EBSD Data from multiple vendors is stored in a data file using the HDF5 file format library. While the general layout of the HDF5 file is the same between vendors there are details that are NOT the same between vendors because each vendors chooses to save different types of data. The top-level datasets that deal with the basic volume information is the same for every file.

Orientations, Reference Frames and Coordinate Systems

DREAM.3D's origin follows the sample coordinate system so that the physical location of the 0 row and 0 column voxel should visually appear in the lower left corner of a computer graphics display. Sample Coordinate System (White) overlaid with EBSD Coordinate System (Yellow).



Commercial EBSD acquisition systems do not typically follow this convention and DREAM.3D needs input from the user so that the proper transformations to the data can be applied during the Reconstruction and other analysis. Commercial EBSD software packages allow for some initial transformations of the data in which case the DREAM.3D environment does not have any way of determining if those transformations have already occured. During the import process the user is asked a few questions regarding the orientation of their EBSD data in relation to the sample coordinate system. Currently there are 3 items that must be answered otherwise undefined behavior may occur during any of the analysis algorithms.

- Some Commercial EBSD acquisition systems allow for a **scan rotation** which rotates the sample coordinate system 180 degrees around the Z axis.
- Should we rotate the Euler angles to bring them in line with the sample reference frame?

HDF5 File Layout

HDF5 Root Level Layout Specification

H5Ebsd Specification		
Root Level Datasets and Group	Attribute Name "FileVersion" Current Value="4"	
Name	**HDF5 Type**	**Value**
Index	H5T_NATIVE_INT32	List of all Slice index values

EulerTransformationAngle	H5T_NATIVE_FLOAT	
EulerTransformationAngle	H5T_NATIVE_FLOAT	
Manufacturer	H5T_STRING	The **Manufacturer** currently is either *TSL* or *HKL*
Max X Points	H5T_NATIVE_INT64	The maximum number of X points in the sample grid
Max Y Points	H5T_NATIVE_INT64	The maximum number of Y points in the sample grid
SampleTransformationAngle	H5T_NATIVE_FLOAT	
SampleTransformationAngle	H5T_NATIVE_FLOAT	
Stacking Order	H5T_NATIVE_UINT32	Defines which slice corresponds to the Z=0 in the coordinate system. Optional Attribute of type H5T_STRING, with Name "Name" and Value ("Low To High" or "High To Low")
X Resolution	H5T_NATIVE_FLOAT	Resolution between sample points in the X direction
Y Resolution	H5T_NATIVE_FLOAT	Resolution between sample points in the Y direction
Z Resolution	H5T_NATIVE_FLOAT	Resolution between slices in the Z Direction
ZEndIndex	H5T_NATIVE_INT64	Starting Slice index
ZStartIndex	H5T_NATIVE_INT64	Ending Slice index (inclusive)
Slice Data organized by Slice index		

Slice Group Specification

Each Slice is grouped into its own H5G_GROUP with the Name of the group simply the index of the slice. Within each slice group are 2 more groups with names **Data** and **Header**

Name	HDF5 Type	Value
Data	H5G_GROUP	Contains all the data columns
Header	H5G_GROUP	Contains all the header entries

TSL Specification

This section details the data to be imported from a .ang file into the .h5ebsd file

TSL (.ang) Data Group Specification

Name	HDF5 Type	Value
Phi1	H5T_NATIVE_FLOAT	Contains all the Phi1 data in a 1D Array with length equal to the total number of points.
Phi	H5T_NATIVE_FLOAT	Contains all the Phi data in a 1D Array with length equal to the total number of points.
Phi2	H5T_NATIVE_FLOAT	Contains all the Phi2 data in a 1D Array with length equal to the total number of points.

X Position	H5T_NATIVE_FLOAT	Contains all the X position data in a 1D Array with length equal to the total number of points.
Y Position	H5T_NATIVE_FLOAT	Contains all the Y position data in a 1D Array with length equal to the total number of points.
Image Quality	H5T_NATIVE_FLOAT	Contains all the Image Quality data in a 1D Array with length equal to the total number of points.
Confidence Index	H5T_NATIVE_FLOAT	Contains all the Confidence Index data in a 1D Array with length equal to the total number of points.
PhaseData	H5T_NATIVE_INT32	Contains all the PhaseData data in a 1D Array with length equal to the total number of points.
SEM Signal	H5T_NATIVE_FLOAT	Contains all the SEM Signal data in a 1D Array with length equal to the total number of points.
Fit	H5T_NATIVE_FLOAT	Contains all the Fit of Solution data in a 1D Array with length equal to the total number of points.

TSL (.ang) Header Group Specification

Name	HDF5 Type	Value
OriginalFile	H5T_STRING	Path to the .ang file that was imported
OriginalHeader	H5T_STRING	Contains the original header from the imported .ctf file
TEM_PIXperUM	H5T_NATIVE_FLOAT	Contains value for the header entry TEM_PIXperUM
x-star	H5T_NATIVE_FLOAT	Contains value for the header entry x-star
y-star	H5T_NATIVE_FLOAT	Contains value for the header entry y-star
z-star	H5T_NATIVE_FLOAT	Contains value for the header entry z-star
WorkingDistance	H5T_NATIVE_FLOAT	Contains value for the header entry WorkingDistance
ElasticConstants	H5T_STRING	Contains value for the header entry ElasticConstants
GRID	H5T_STRING	Contains value for the header entry GRID
XSTEP	H5T_NATIVE_FLOAT	Contains value for the header entry XSTEP
YSTEP	H5T_NATIVE_FLOAT	Contains value for the header entry YSTEP
NCOLS_ODD	H5T_NATIVE_INT32	Contains value for the header entry NCOLS_ODD
NCOLS_EVEN	H5T_NATIVE_INT32	Contains value for the header entry NCOLS_EVEN
NROWS	H5T_NATIVE_INT32	Contains value for the header entry NROWS

OPERATOR	H5T_STRING	Contains value for the header entry OPERATOR
SAMPLEID	H5T_STRING	Contains value for the header entry SAMPLEID
SCANID	H5T_STRING	Contains value for the header entry SCANID
Phases	H5G_GROUP	Group that contains a subgroup for each phase where the name of each subgroup is simply the index of the phase starting at 1.

TSL (.ang) Phase Group Specification

Name	HDF5 Type	Value
Categories	H5T_NATIVE_INT32	
Formula	H5T_STRING	
Info	H5T_STRING	
LatticeConstants	H5T_NATIVE_FLOAT	
LatticeConstants	H5T_STRING	
NumberFamilies	H5T_NATIVE_INT32	
Phase	H5T_NATIVE_INT32	
Symmetry	H5T_NATIVE_INT32	
hklFamilies	H5G_GROUP	Contains all the HKL Family information where the number of datasetscontained in this group is the number of HKL Families

TSL (.ang) HKLFamily Group Specification

Name	HDF5 Type	Value
{Based on Index of the family}. If there are 4 families then there are 4 data sets with names "0", "1", "2" and "3".	Custom: See Below	

```
typedef struct
{
    int h;@n
    int k;@n
    int l;@n
    int s1;@n
    float diffractionIntensity;@n
    int s2;@n
} HKLFamily_t;
```

HKL Specification

This section details the data to be imported from a .ctf file into the .h5ebsd file

HKL (.ctf) Data Group Specification

1D Array of Values where the number of elements in the array is equal to the total number of points per 2D Slice.

Name	HDF5 Type	Value

Phase	H5T_NATIVE_INT32	
X	H5T_NATIVE_FLOAT	
Y	H5T_NATIVE_FLOAT	
Z	H5T_NATIVE_FLOAT	Note that this ONLY appears in a "3D" .ctf data file
Bands	H5T_NATIVE_INT32	
Error	H5T_NATIVE_INT32	
Euler1	H5T_NATIVE_FLOAT	In 2D files these are in Degrees. In 3D files these are in Radians
Euler2	H5T_NATIVE_FLOAT	In 2D files these are in Degrees. In 3D files these are in Radians
Euler3	H5T_NATIVE_FLOAT	In 2D files these are in Degrees. In 3D files these are in Radians
MAD	H5T_NATIVE_FLOAT	
BD	H5T_NATIVE_INT32	
BS	H5T_NATIVE_INT32	
GrainIndex	H5T_NATIVE_INT32	
GrainRandomColourR	H5T_NATIVE_UINT8	
GrainRandomColourG	H5T_NATIVE_UINT8	
GrainRandomColourB	H5T_NATIVE_UINT8	

HKL (.ctf) Header Specification

Name	HDF5 Type	Value
OriginalFile	H5T_STRING	Path to the .ctf file that was imported
OriginalHeader	H5T_STRING	Contains the original header from the imported .ctf file
Prj	H5T_STRING	Contains value for the header entry Prj
Author	H5T_STRING	Contains value for the header entry Author
JobMode	H5T_STRING	Contains value for the header entry JobMode
XCells	H5T_NATIVE_INT32	Contains value for the header entry XCells
YCells	H5T_NATIVE_INT32	Contains value for the header entry YCells
XStep	H5T_NATIVE_FLOAT	Contains value for the header entry XStep
YStep	H5T_NATIVE_FLOAT	Contains value for the header entry YStep
ZStep	H5T_NATIVE_FLOAT	Contains value for the header entry ZStep
ZCells	H5T_NATIVE_FLOAT	Contains value for the header entry ZCells
AcqE1	H5T_NATIVE_FLOAT	Contains value for the header entry AcqE1
AcqE2	H5T_NATIVE_FLOAT	Contains value for the header entry AcqE2
AcqE3	H5T_NATIVE_FLOAT	Contains value for the header entry AcqE3

Mag	H5T_NATIVE_INT32	Contains value for the header entry Mag
Coverage	H5T_NATIVE_INT32	Contains value for the header entry Coverage
Device	H5T_NATIVE_INT32	Contains value for the header entry Device
KV	H5T_NATIVE_INT32	Contains value for the header entry KV
TiltAngle	H5T_NATIVE_FLOAT	Contains value for the header entry TiltAngle
TiltAxis	H5T_NATIVE_FLOAT	Contains value for the header entry TiltAxis
Phases	H5G_GROUP	Group that contains a subgroup for each phase where the name of each subgroup is simply the index of the phase starting at **1.**

HKL (.ctf) Phase Group Specification

Name	HDF5 Type	Value
Comment	H5T_STRING	Contains value for the header entry Comment
Internal1	H5T_STRING	Contains value for the header entry Internal1
Internal2	H5T_STRING	Contains value for the header entry Internal2
LatticeAngles	H5T_NATIVE_FLOAT	Contains value for the header entry Lattice Angles in a 1x3 array
LatticeDimensions	H5T_NATIVE_FLOAT	Contains value for the header entry Lattice Dimensions in a 1x3 array
LaueGroup	H5T_NATIVE_INT32	Contains value for the header entry LaueGroup plus an H5T_STRING Attribute which is the string name of the Laue Group for example "Hexagonal-High 6/mmm"
SpaceGroup	H5T_NATIVE_INT32	Contains value for the header entry SpaceGroup
PhaseName	H5T_STRING	Contains value for the header entry PhaseName

Stacking Order Discussion

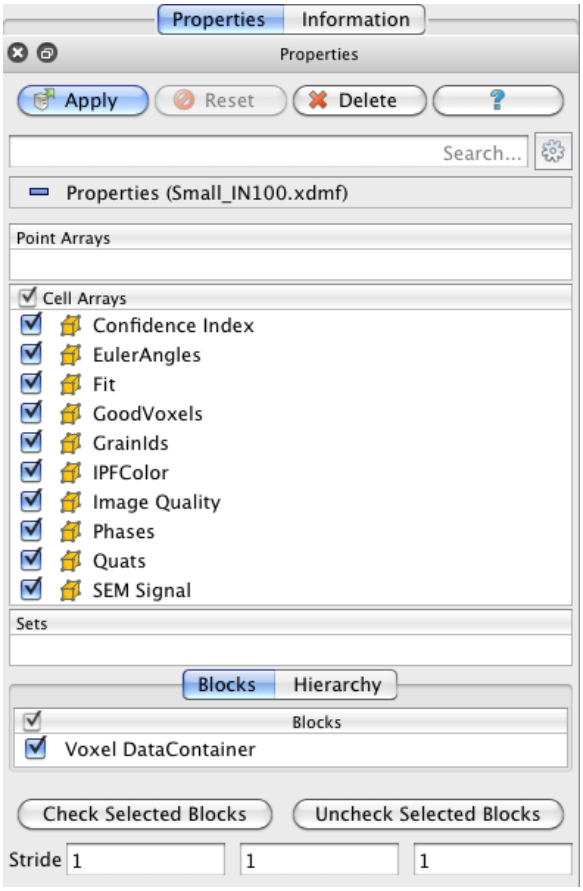
The **Stacking Order** refers to the order in which the z slices are stacked together when they are read from the file. The enumerations are also in the EbsdLibConstants.h header file.

As a further explanation if the ordering is **Low To High** then the slice with the lowest number is positioned at Z=0 in 3D Cartesian space. For example if your data set is numbered from 23 to 86 with file names of the form Slice_-023.ang and you select "Low To High" then the data inside of file Slice_023.ang will be positioned at Z=0 during any method that has to deal with the data. The opposite of this is if the user were to select to have their data **High to Low** in which case the file with name Slice_086.ang will be positioned at Z=0 and the file with name "Slice_023.ang" will be positioned at Z=64.

7.3 XDMF For Visualization

Introduction

XDMF is an XML file that describes either a regular grid or unstructured grid data set where the data can be stored in an HDF5, Binary or with the xml file itself. The user is can find out more information for this format at the Xdmf website. DREAM3D will optionally write out an Xdmf file when writing a DREAM3D Data file so that users can view the processed data easily using the free ParaView application. In prior versions of DREAM3D the only way to view the data was to write out a legacy VTK file which can be slow and simply duplicates much of the data that is stored in the .dream3d file. This is a waste of space and not memory efficient when it comes to ParaView loading the data from the legacy .vtk file.



By using an Xdmf wrapper file the user can selectively load just the data that they are interested in viewing. This saves time loading the data and more importantly reduces the amount of memory needed to load, store and view the data.

Note: It is a REQUIREMENT that the Xdmf file and the DREAM3D file are in the same directory because paths in the XDMF file refer to data stored in the DREAM3D file. The XML files are typically a few kilobytes in size and are therefor a huge savings over writing the VTK file for the same data.

XDMF Example Contents

Below is an example of what an Xdmf file might contain.

```
<?xml version="1.0"?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd"[]>
<Xdmf xmlns:xi="http://www.w3.org/2003/XInclude" Version="2.2">
 <Domain>
  <Grid Name="Voxel DataContainer" GridType="Uniform">
    <Topology TopologyType="3DCoRectMesh" Dimensions="183 885 1025 "></Topology>
    <Geometry Type="ORIGIN_DXDYDZ">

      <DataItem Format="XML" Dimensions="3">0 0 0</DataItem>

      <DataItem Format="XML" Dimensions="3">1 1 1</DataItem>
```

```
    </Geometry>
    <Attribute Name="ImageData" AttributeType="Scalar" Center="Cell">
      <DataItem Format="HDF" Dimensions="182 884 1024 " NumberType="UChar" Precision="1" >
        out.dream3d:/VoxelDataContainer/CELL_DATA/ImageData
      </DataItem>
    </Attribute>
  </Grid>
 </Domain>
</Xdmf>
```

7.4 Import & Export File Formats

Supported Input File Types

- TSL .ang files
- HKL .ctf files
- HEDM .mic files
- Raw Binary Files
- Dx files (Limited Subset)
- H5Ebsd files
- Ph files
- Vtk file (Structured Points)
- DREAM3D Native Files

Supported Output File Types

- Avizo Rectilinear Coordinate
- Avizo Uniform Coordinate Writer
- H5EBSD files
- DREAM3D Native files
- Dx files (Limited Subset)
- Ph files
- SPParks file
- STL Files (StereoLithography Format)
- VTK Rectilinear Grid (Voxel Based Data)
- VTK Polydata (Surfacemesh Based Data)
- VTK Polydata (Solidmesh Based Data)
- INL (CSV File of Voxel attributes)
- Nodes/Triangles from Surfacemesh
- CSV File of Field Values

Please take a look at each of the various filters that read or write those files for notes on each type of file.

CHAPTER 8

Tutorials

DREAM3D is capable of a wide variety of reconstruction, data analysis and synthetic microstructure which can present issues to the new user with regards to how to get started constructing a working pipeline. In these sections a few sample pipelines are presented which will allow the new user to get started quickly.

3D Orientation Data Reconstruction This tutorial describes how to reconstruct EBSD data into a 3D Voxel based volume and visualize the results with ParaView.

Synthetic Microstructure Generation This tutorial shows the user how to create a pipeline that will generate a single phase equiaxed synthetic microstructure from a set of statistics generated with StatsGenerator

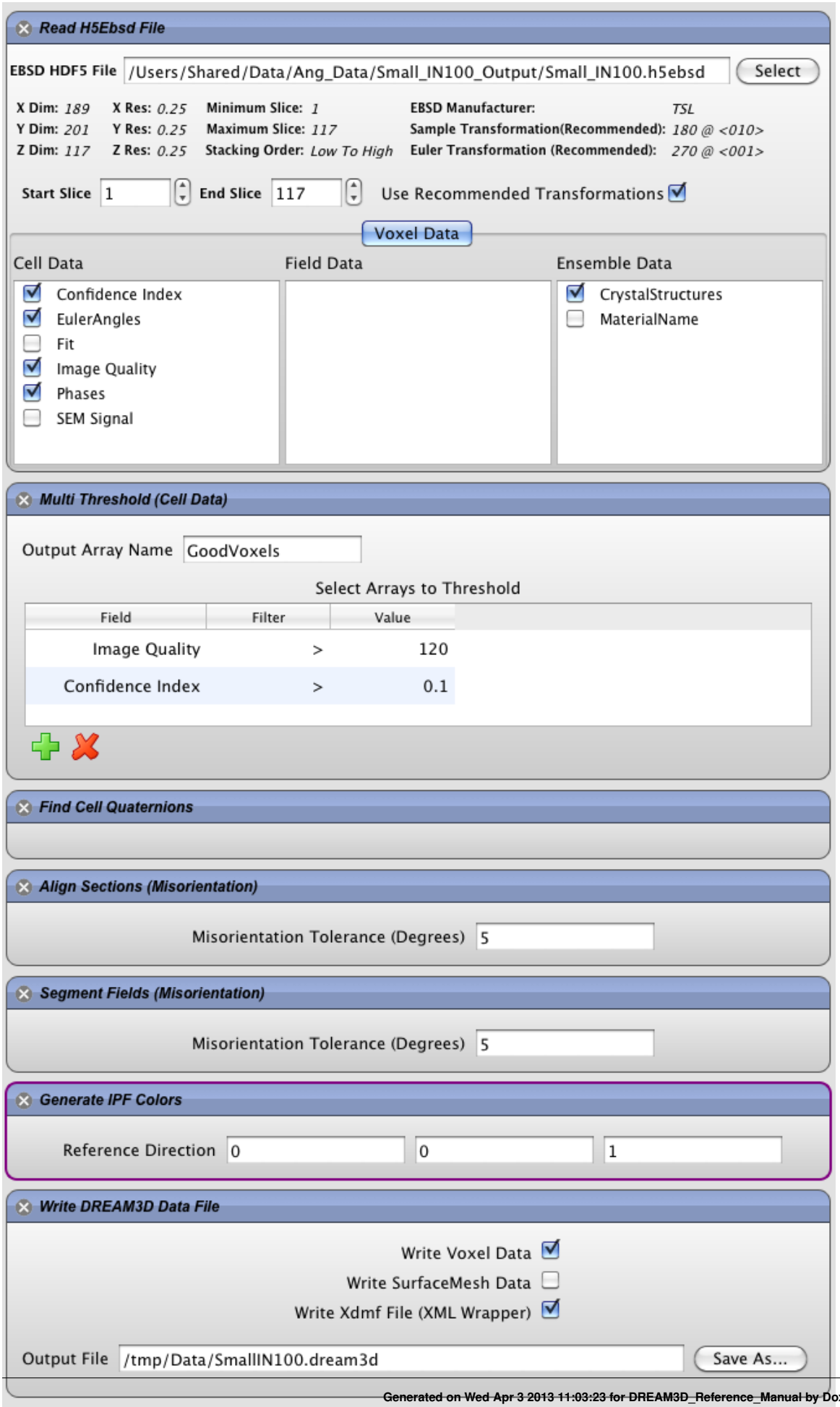
Surface Meshing Tutorial This tutorial serves as an introduction to creating a triangle based surface mesh from a 3D voxel volume. Background information on the terminology used in the filters is also presented.

Cropping Data Tutorial This tutorial shows how to use ParaView to crop volumes of data using the "Crop Volume" filter.

8.1 3D Orientation Data Reconstruction

With the advent of 3D EBSD data collection systems the visualization of the collected data is typically the first order of business. Using the sample data set *Small IN100* which is a Nickel based super alloy we can use DREAM3D to build a 3D reconstruction of the 2D slices. The order of operations for this type of work flow is the following:

1. `Convert the Orientation Data` to an H5Ebsd based archive file.
2. `Read H5Ebsd file`.
3. `Determine which voxels are considered good`.
4. `Find the Quaternions for each Cell`.
5. `Align the sections` using a Misorientation Alignment filter
6. `Segment the grains` using a Misorientation Algorithm.
7. `Generate the IPF` (Inverse Pole Figure) colors based on the Euler Angles.
8. `Write out a DREAM3D` data file (with optional Xdmf file).


Read H5Ebsd File
EBSD HDF5 File
/Users/Shared/Data/Ang_Data/Small_IN100_Output/Small_IN100.h5ebsd
Select
X Dim: 189
X Res: 0.25
Minimum Slice: 1
EBSD Manufacturer:
TSL
Y Dim: 201
Y Res: 0.25
Maximum Slice: 117
Sample Transformation(Recommended):
180 @ <010>
Z Dim: 117
Z Res: 0.25
Stacking Order: Low To High
Euler Transformation (Recommended):
270 @ <001>
Start Slice
1
End Slice
117
Use Recommended Transformations
Voxel Data
Cell Data
Field Data
Ensemble Data
Confidence Index
EulerAngles
Fit
Image Quality
Phases
SEM Signal
CrystalStructures
MaterialName
Multi Threshold (Cell Data)
Output Array Name
GoodVoxels
Select Arrays to Threshold
Field
Filter
Value
Image Quality
>
120
Confidence Index
>
0.1
Find Cell Quaternions
Align Sections (Misorientation)
Misorientation Tolerance (Degrees)
5
Segment Fields (Misorientation)
Misorientation Tolerance (Degrees)
5
Generate IPF Colors
Reference Direction
0
0
1
Write DREAM3D Data File
Write Voxel Data
Write SurfaceMesh Data
Write Xdmf File (XML Wrapper)
Output File
/tmp/Data/SmallIN100.dream3d
Save As...

After running the pipeline `ParaView` can be used to display the generated Xdmf file. There are a few options within ParaView that need to be set in order to show the encoded IPF colors otherwise ParaView will create a random color table for the visualization.

1. Set the representation to *Surface*
2. Set the color by to *IPFColor*
3. Click the little gear icon to show the advanced options
4. Uncheck the **Map Scalars** and **Interpolate Scalars**.

After all of these options are set the visualized volume will display with the encoded IPF Colors. The user is free to use what ever coloring scheme they would like for example coloring by the phase or Grain Id will also show grain boundaries.

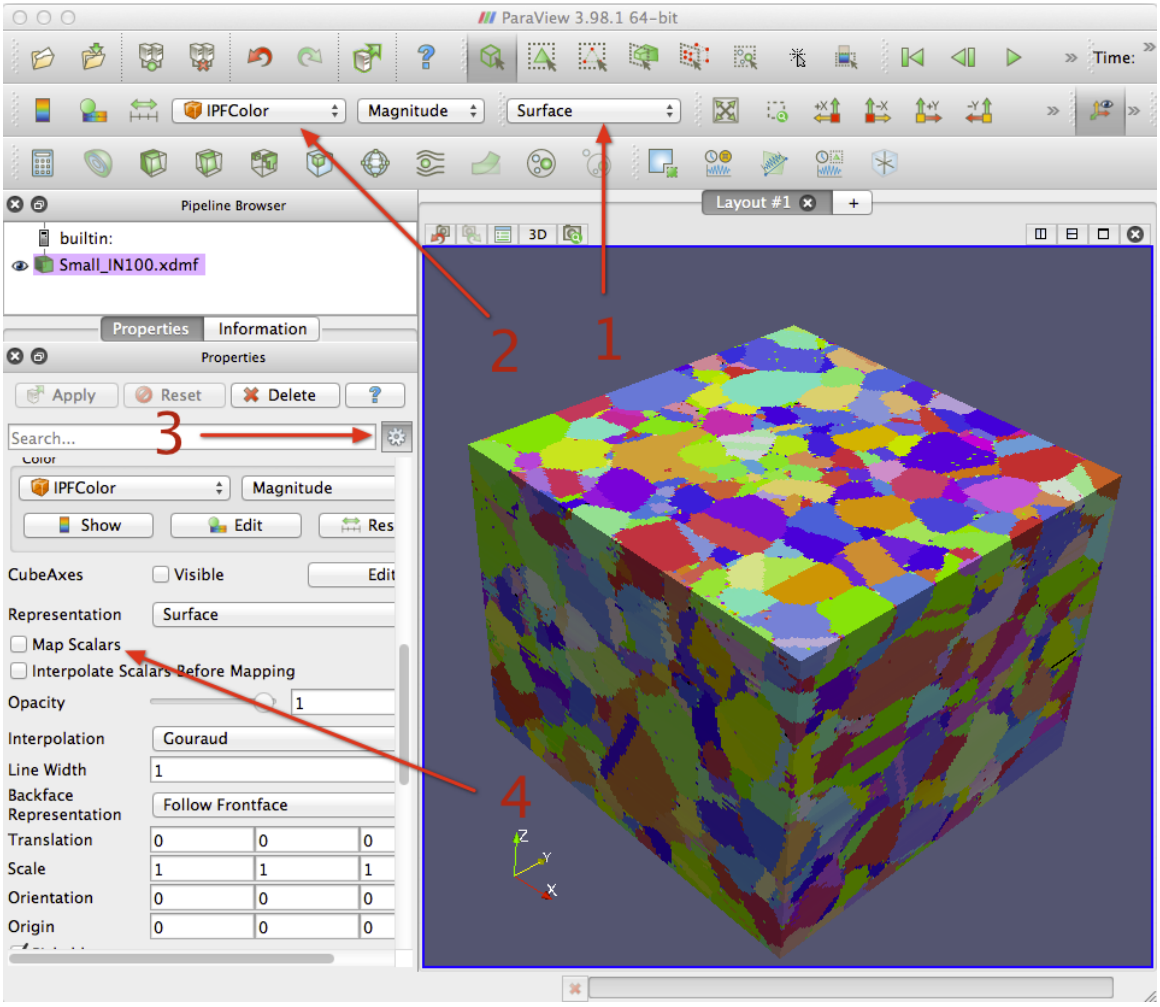


Figure 8.1: 3D Reconstruction of EBSD Orientation Data

8.2 Synthetic Microstructure Generation

Launch DREAM3D and create the pipeline that is displayed in the following image.

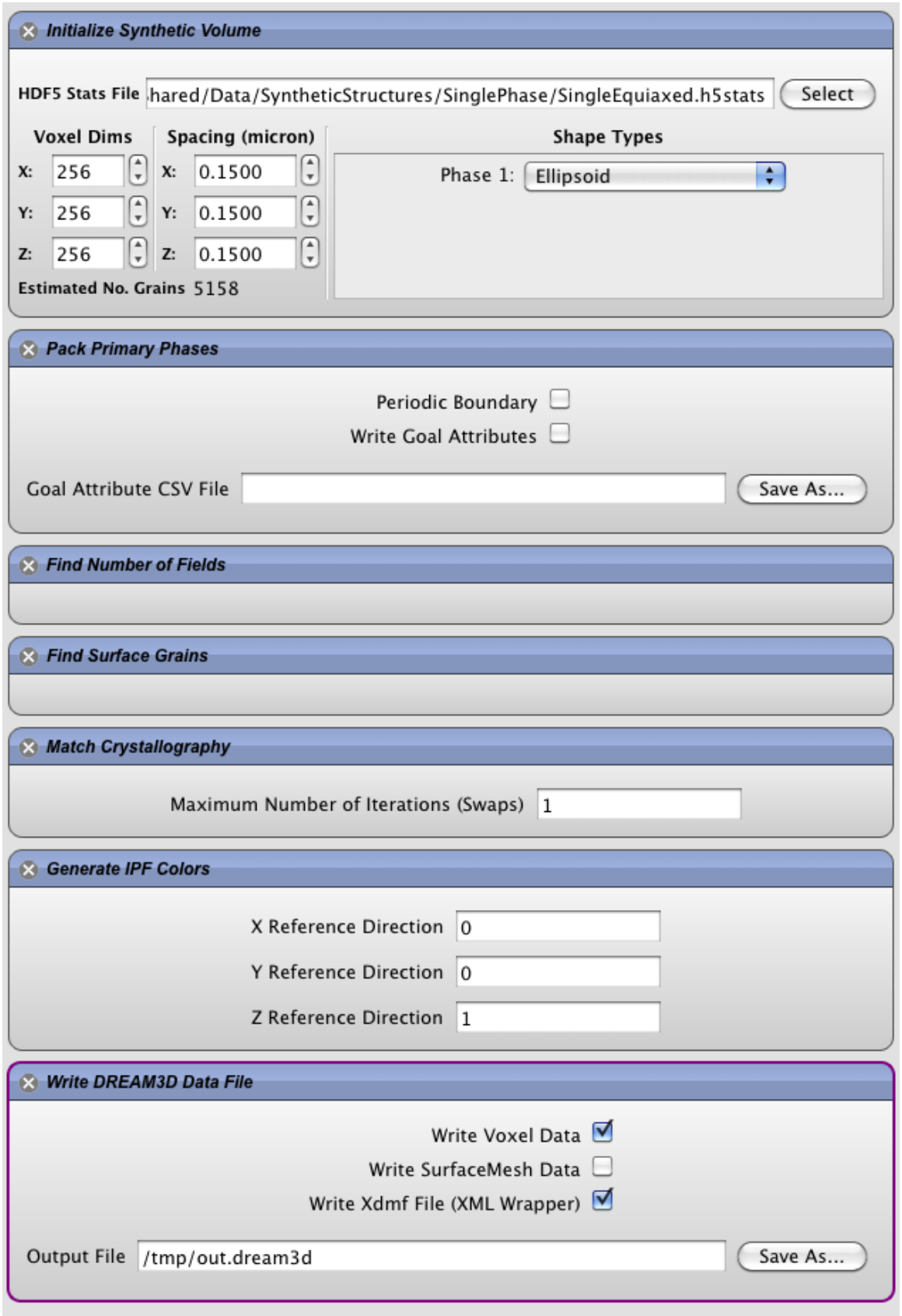


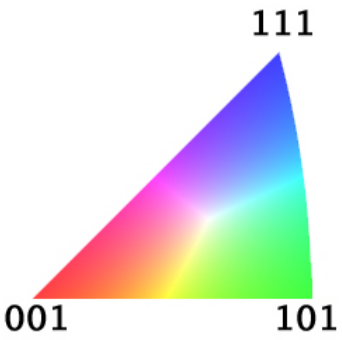
Figure 8.2: DREAM3D Gui showing complete pipeline

Note that you need to fill out the Voxel Dims and the Spacing with NON-Zero values in order for the generation to

complete successfully.

Pipeline

- Initialize Synthetic Volume
  - 256 Voxels in X,Y & Z
  - 0.15 micron resolution in X, Y & Z
- Pack Primary Phases
- Find Number of Fields
- Find Surface Grains
- Match Crystallography
- Generate IPF Colors (Optional)
  - This filter is Optional
  - It will generate actual colors corresponding to the standard IPF Color Triangle
- Write DREAM3D Data File



Notes

Be sure to properly select the .h5stats (or .dream3d) input file and properly select an output file to save the data. We will save the data as a .dream3d file with the XDMF file wrapper enabled. **Make sure you do NOT name the output file the same as the input file or it will get over written**.

Visualization

From ParaView open the .xdmf file and select the "Grain Ids" and "IPFColors" data sets then click the "Apply" button. The microstructure should look something like below which is shown with IPF coloring with a 001 reference direction.

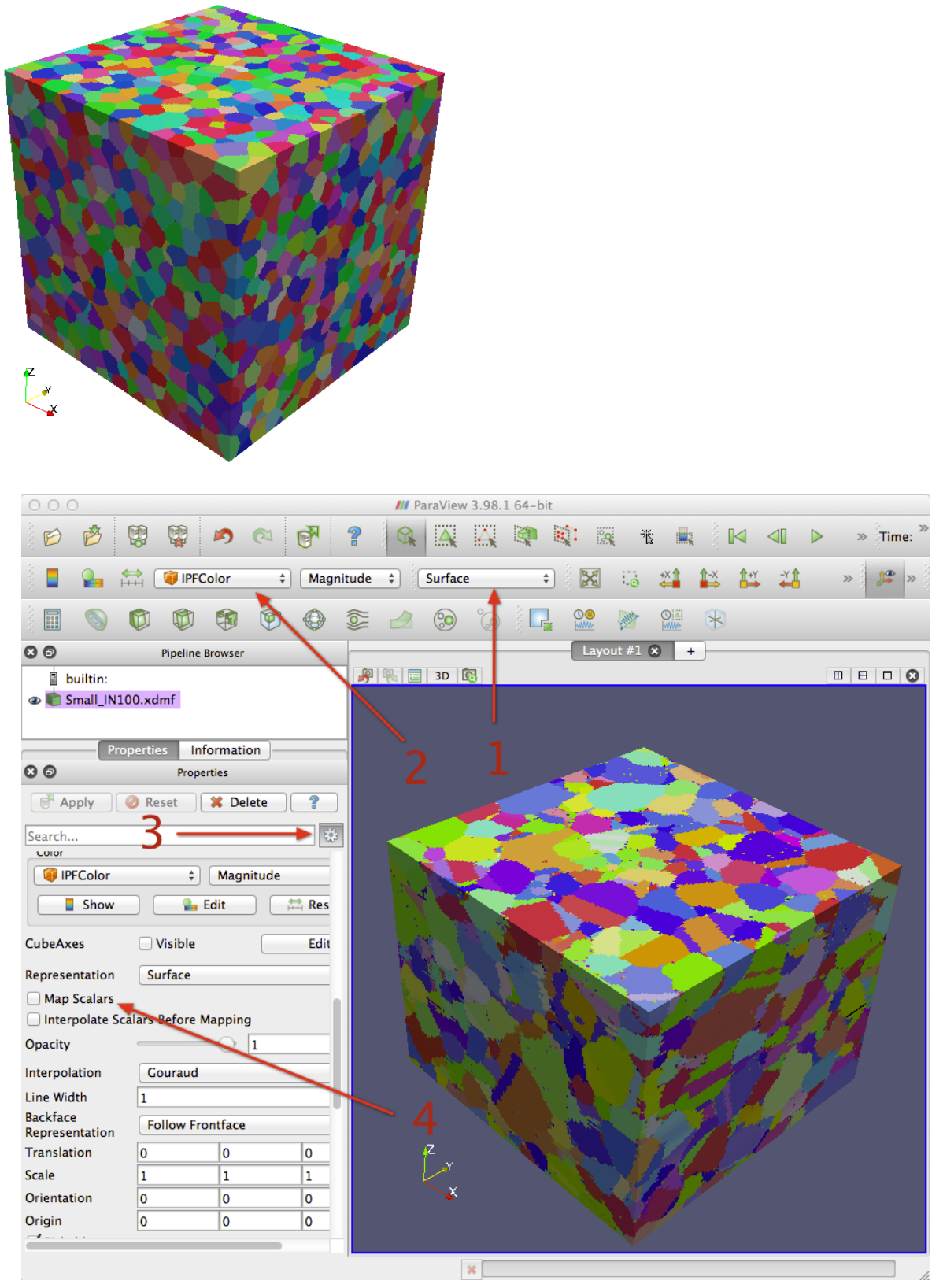
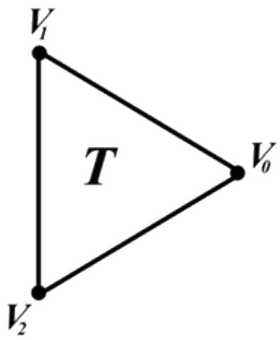


Figure 8.3: ParaView OverView

8.3 Surface Meshing Tutorial

Surface Mesh Definitions

Creating a surface mesh in DREAM3D can be accomplished in several ways. There are a pair of filters `Multi-Material Marching Cubes (Slice at a Time)` and `Quick Surface Mesh` that will generate a triangular based mesh of the primary **Field** for you. The terminology of the underlying data structures is something that the user should understand so that they can be well informed. We start out by defining a triangle **T** which we will call a **Face**. Each Triangle has 3 **Vertex** (V0, V1, V2) which are referred to as a **Point**, **Vertex** and/or **Node** interchangeably within DREAM3D.



These data structures are declared in DREAM3D in the following code:

```
typedef struct
{
    Float_t pos[3];
} Vert_t;

typedef struct
{
    signed int verts[3];
} Face_t;
```

DREAM3D stores all the vertices in a single array. DREAM3D has the notion of a "shared vertex list" where each vertex is only listed once in the vertex list. Each Triangle is then created by referencing 3 indexes out of the vertex list to form the 3 points of the triangle.

Vertex & Triangle Attributes

DREAM3D filters can attach *attributes* to each vertex or triangle. These attributes can be *scalar* or *vector* of any numerical type. Some attributes are dependent on the winding of the triangle. From OpenGL the normal is computed via a right-hand-rule and so proper triangle winding is important when visualizing the mesh and also is important to some filters such as the `Grain Face Curvature Filter`. This is why there are filters such as `Reverse Triangle Winding` and `Verify Triangle Winding` in order to ensure proper winding and normal calculations.

Surface Mesh Data Container

To support the creation and manipulation of a surface mesh DREAM3D introduced a new data container type called the `SurfaceMeshDataContainer` and have as its core properties the shared vertex array and triangle array. These arrays have default names given to them which are defined in `DREAM3DLib/Common/Constants.h`

```
const std::string SurfaceMeshNodes("SurfaceMeshNodes");
const std::string SurfaceMeshTriangles("SurfaceMeshTriangles");
```

In order to stay consistent with the VoxelDataContainer and its terminology DREAM3D refers to *Vertex Attributes* as **Cell** data and *Triangle Attributes* as **Field** data. There are also convenience methods that are named `setPointData(...)`, `setFaceData(...)` and `setEdgeData(...)` that map loosely to the **Cell**, **Field** and **Ensemble** syntax of the `VoxelDataContainer`

In the filter documentation the following terminology will be used:

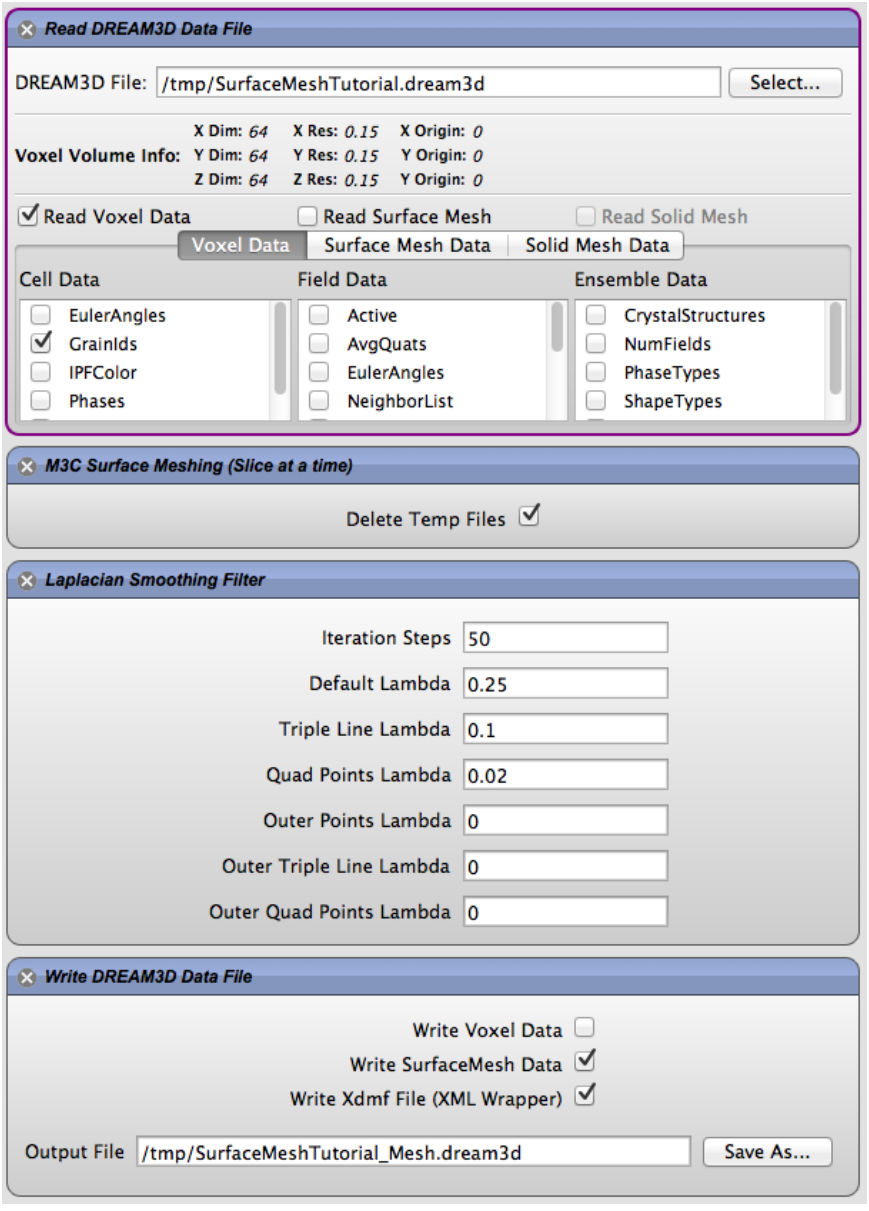
- Vertex Array: Refers to the shared vertex array stored in the `SurfaceMeshDataContainer`
- Triangle Array: Refers to the triangle array stored in the `SurfaceMeshDataContainer`
- Edge Array: Refers to the edge array stored in the `SurfaceMeshDataContainer`

Edge Data

Update with algorithm to generate Unique Edge Ids

Surface Meshing Tutorial

For this tutorial we are going to be using a pre-made DREAM3D data file. The file is simply a Single Phase Synthetic Structure that was quickly generated using the Synthetic Microstructure Creation abilities of DREAM3D. There are other options that exist to create a surface mesh. One can read in a Ph or Dx file, use the Raw Binary Reader to read in the Raw Grain Ids from a binary file or have an already existing set of Grain Ids from a DREAM3D pipeline that segmented the grains for you.



Start by adding the "Read DREAM3D Data File" filter. Select the file that you just downloaded an then only check the "GrainIds" data under the "Cell Data" section. This will ensure that this is the only data that is loaded from the file. Since creating the mesh can take large amounts of memory we only want to load the data necessary.

Next add in the "M3C Surface Meshing (Slice at a time)" filter which will create the actual surface mesh. Leave the option to "Delete Temp Files" checked on. The files are in a binary format meant for debugging if things go wrong during the meshing.

After we get the surface mesh we could simply write out a DREAM3D file or a VTK legacy file but we are selecting to also smooth the surface mesh to prepare it for other analysis.

In order to smooth the mesh we now add the "Laplacian Smoothing Filter" which takes 7 arguments. The first argument is the number of iterations of smoothing to use. The remaining 6 arguments are 2 sets of 3 arguments where you can set the lambda value for the bulk nodes, triple line nodes and quad point nodes for both the "inside triangles" and those triangles that touch the outer surface of the model. The user should understand that a side effect of the Laplacian smoothing algorithm is shrinkage of the actual value due to movement of the nodes. If enough iterations are run and all 6 arguments have valid value (0 to 1) then the volume can collapse on itself. This is why we have the 6 arguments to alleviate some of the shrinkage by pinning the outer shell of triangles in place and only allowing the internal triangles to move. We can also allow the bulk triangles to move more than the triple line or quad point nodes.

Finally after the Smoothing filter add a "Write DREAM3D Data File" filter. Be sure to check the option to write the SurfaceMesh Data and also make sure the "Write Xdmf File" is also checked. After smoothing the mesh can be viewed with ParaView as seen in the image below. See the DREAM3D documentation for more details on the exact storage layout of the mesh within the DREAM3D data file.

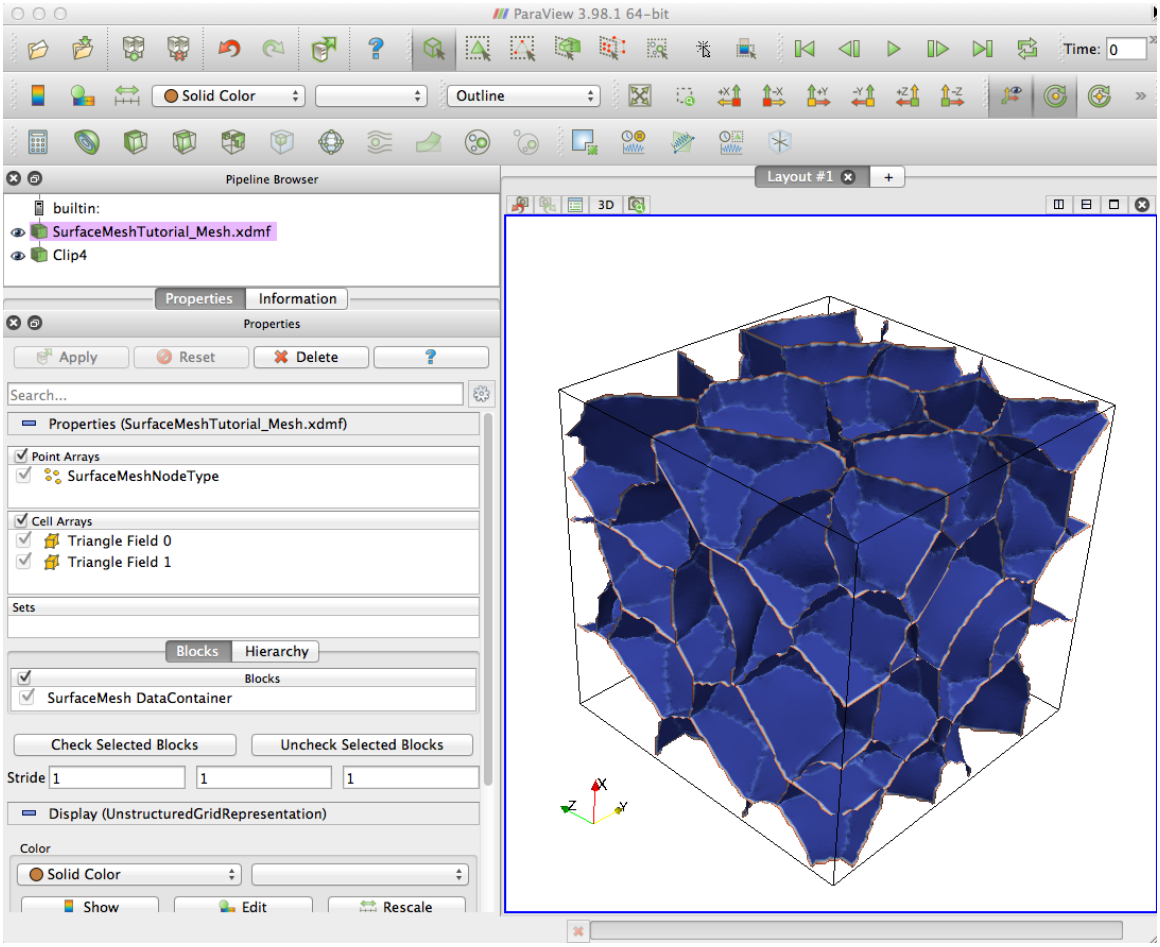
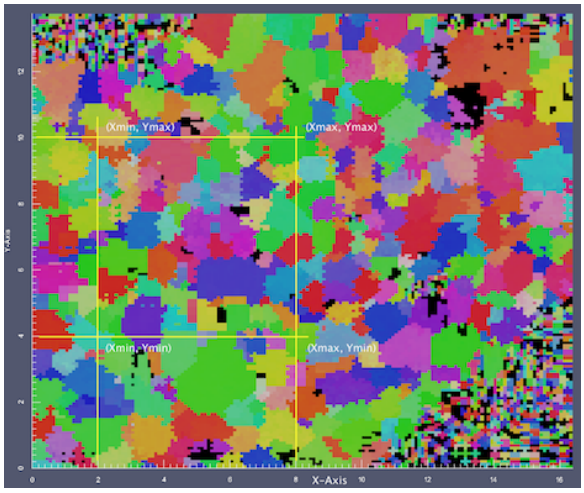


Figure 8.4: Mesh Generated from DREAM3D

8.4 Cropping Data Tutorial

Cropping data in DREAM3D can help reduce the time to process data by removing data that the user decides should not be processed. The filter uses the 3D Voxel index as the minimum and maximum values to use for the region of interest. The easiest way to work through the data is to initially run a pipeline that writes out a .dream3d file and load the .xdmf file into ParaView. Then in the *Information* tab ParaView will display the extents of the data. For this example note that the data extent in Voxels is (0,164)(0,172)(0,39) and in Microns is (0, 16.4)(0, 13.8)(0, 9.75) which corresponds to a resolution of (0.1, 0.08, 0.25).



The display in ParaView shows the values in Microns so we need to convert those to Voxels (Or just estimate if that is good enough). Again, for this example we want to crop out a section from (2, 8) in X and (4, 10) in Y and all of Z.

Extents
X Extent: 0 to 164 (dimension: 165)
Y Extent: 0 to 172 (dimension: 173)
Z Extent 0 to 39 (dimension: 40)
Bounds
X range: 0 to 16.4 (delta: 16.4)
Y range: 0 to 13.8 (delta: 13.8)
Z range: 0 to 9.75 (delta: 9.75)

The general formula to figure this out is the following.

```
Micron * Resolution(Voxel/Micron) = Voxel
```

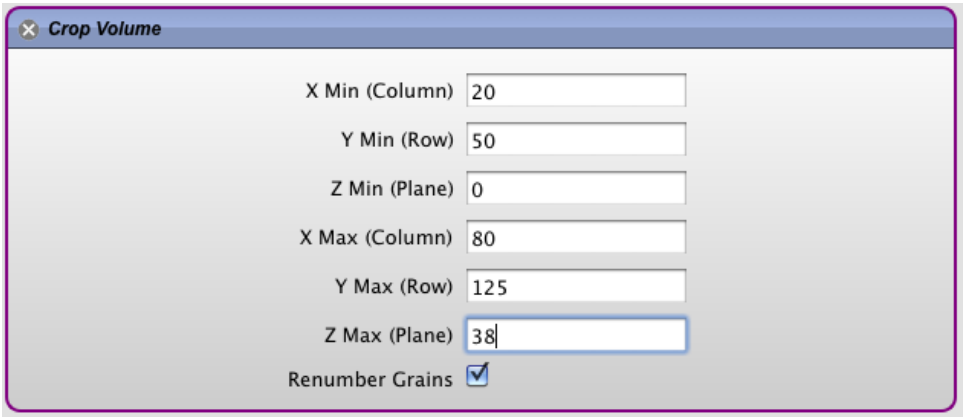
Applying this to our sample data gives the following table.

Example Calculation for Min/Max Voxels for Cropping

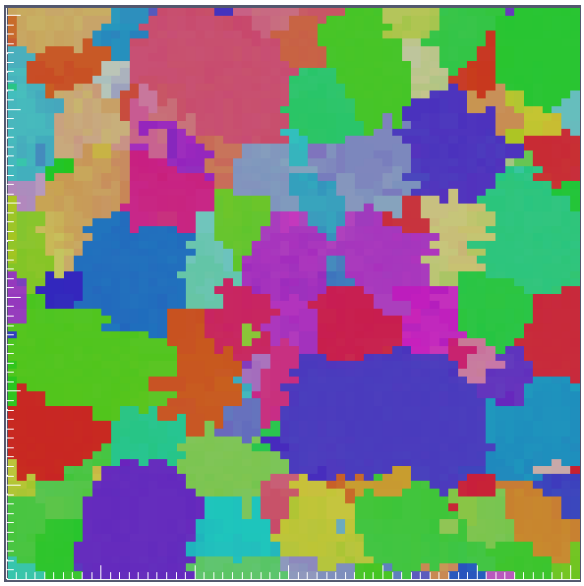
Axis	Min	Max
X	2 / 0.1 = 20	8 / 0.1 = 80
Y	4 / 0.08 = 50	10 / 0.08 = 125
Z	0 / 0.25 = 0	9.75 / 0.25 = 39 -1

We subtract 1 from the max Z value because the internal arrays are Zero Based.

Entering these values into the "Crop Volume" filter gives us the following:



After the filter executes the data from the volume of interest is cropped out.



CHAPTER 9

Auxiliary Tools

Stats Generator

A discussion of the helper tool for generating uer defined statistics for building synthetic structures in DREAM.3D.

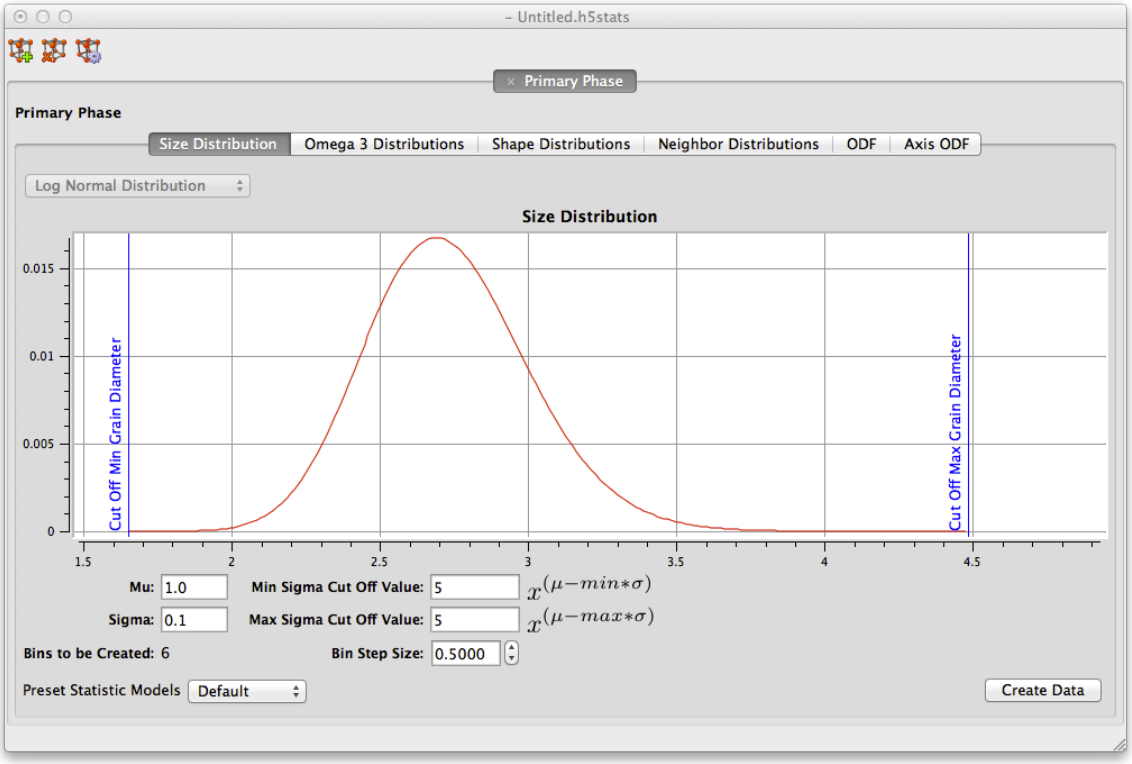
9.1 Stats Generator

Introduction

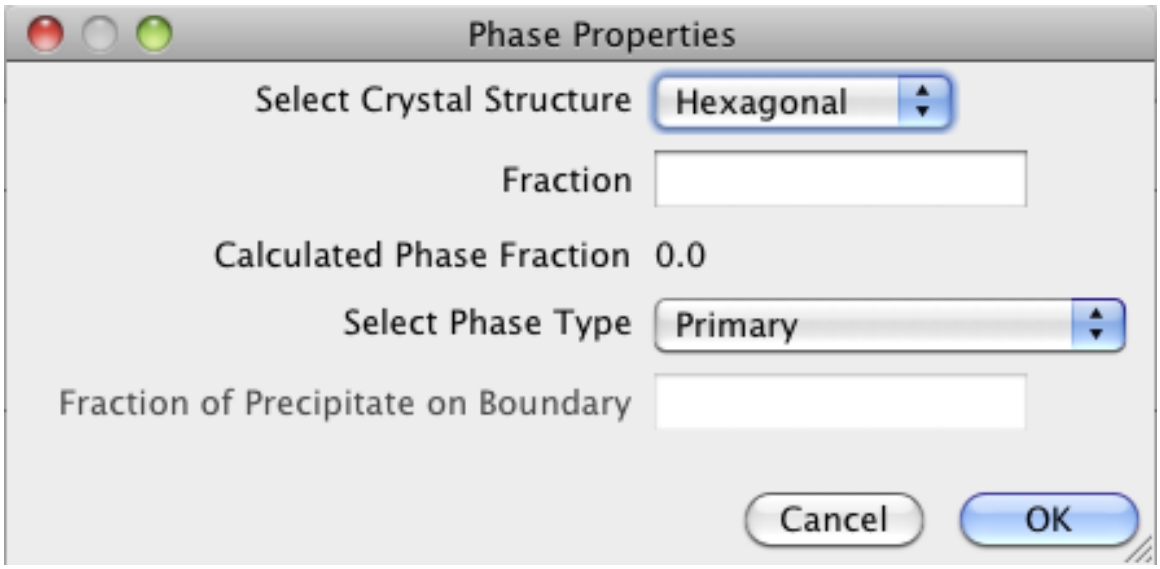
In order to allow the user to quickly generate a set of statistics that can be used to create a synthetic microstructure the “StatsGenerator” program was created. Using this program the user can generate the necessary statistics that describe a microstructure and save those statistics to a DREAM3D file that can then be used in a DREAM3D pipeline to generate a synthetic microstructure. The main features are:

- Describe the grain size distribution via a simple Gaussian distribution
- Select from several preset microstructure types
- Define the Omega 3 Distributions
- Define the Shape distributions
- Define the Neighbor distributions
- Define a Crystallographic Texture through ODF parameters
- Define the MDF through various parameters
- Define the Axis ODF for the grains
- Add as many phases as the user needs
- Define each phase according to several presets
  - Primary
  - Precipitate
  - Transformation
  - Matrix
  - Boundary

Size Distribution Tab



- **Mu** This is the average value of the lognormal grain size distribution
- **Sigma** This is the standard deviation of the lognormal grain size distribution
- **Sigma Cut Off Value** This allows the user to truncate the distribution to remove very large grains
- **Bin Step Size** This is the size of bin to use in segregating the grains into size classes for correlating other statistics to grain size. Note that the Bins to be Created is displayed in the bottom left corner.
- **Preset Statistic Models** This allows the user to select a *morphological-type* of microstructure to populate the default data.
  - Default: this populates the statistic tabs with generic random data that may not create a ÔbuildableÕ microstructure. This option should only be used if the user is going to enter the values on all the statistics tabs themselves.
  - Equiaxed: this populates the statistic tabs with data that is designed to generate a random equiaxed microstructure.
  - Rolled: this populates the statistic tabs with data that is designed to generate a rolled microstructure with elongated grains with user defined aspect ratios.
    - Aspect Ratio 1: this is the aspect ratio between the dimensions of the grains in the rolling direction and transverse direction, respectively.
    - Aspect Ratio 2: this is the aspect ratio between the dimensions of the grains in the rolling direction and normal direction, respectively. This value must be larger than Aspect Ratio 1 or the default values will be wrong.
- **Phase Properties** this is the area where the user can enter information about the phase for which statistics are currently being generated. The Plus Button allows the user to add a phase, the Minus Button allows the user to remove a phase and the Wheel Button allows the user to edit the currently selected phase.



- Select Crystal Structure: this allows the user to specify the crystal structure of the phase. Currently, the two options are Cubic and Hexagonal.

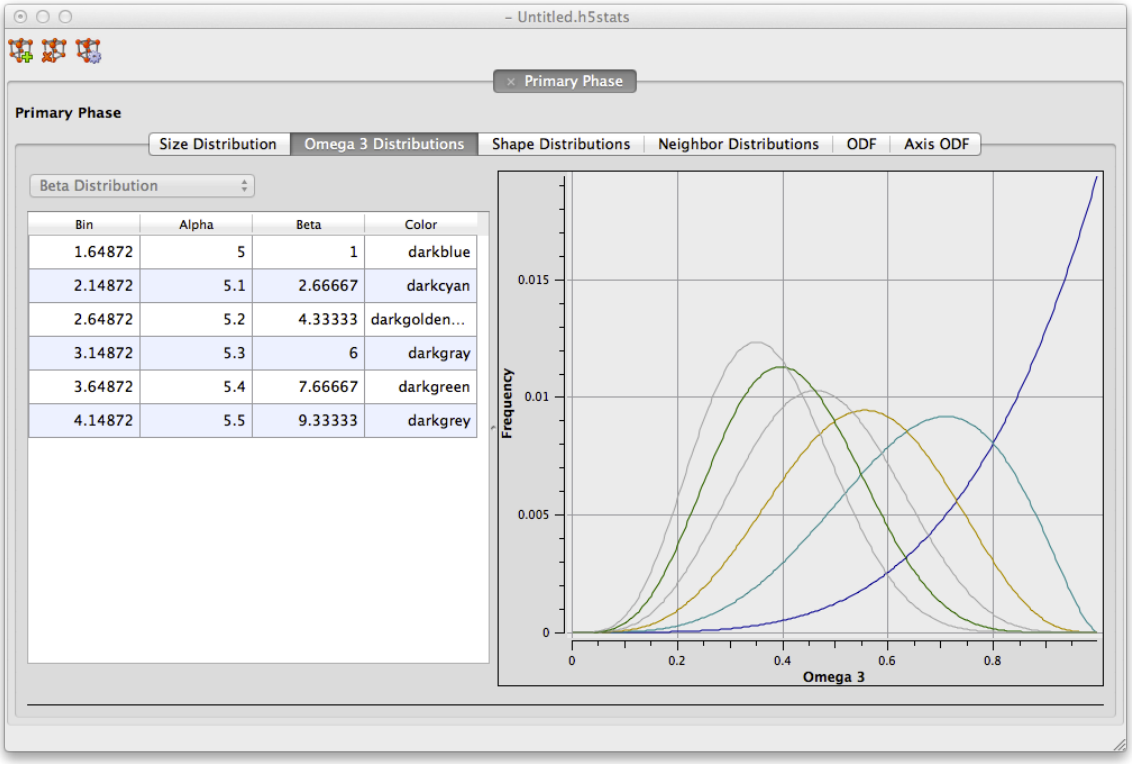
- Fraction: this is the volume fraction of the phase. The Calculated Phase Fraction is updated as more phases are added, by scaling the current total of all the phases fractions to 1 (in case the user's total is not equal to 1 when finished).

- Select Phase Type: this specifies the type of the phase. Currently, the two types of phases are Primary and Precipitate. Note that there must be at least one Primary phase before any Precipitate phase can be created or the Synthetic Builder will fail.

- Fraction of Precipitate on Boundary: if the phase type is set to Precipitate, then the user must specify the number fraction of the precipitates that are located on grain boundaries of the primary phase. This value will be scaled to 1 if the userÕs value is larger than 1. The value is keyed to -1 for Primary phases.

- **Create Data** This locks in the values the user has entered and populates the other tabs with default values. The user can move through the other tabs and change any default values if desired.

Omega3 Tab



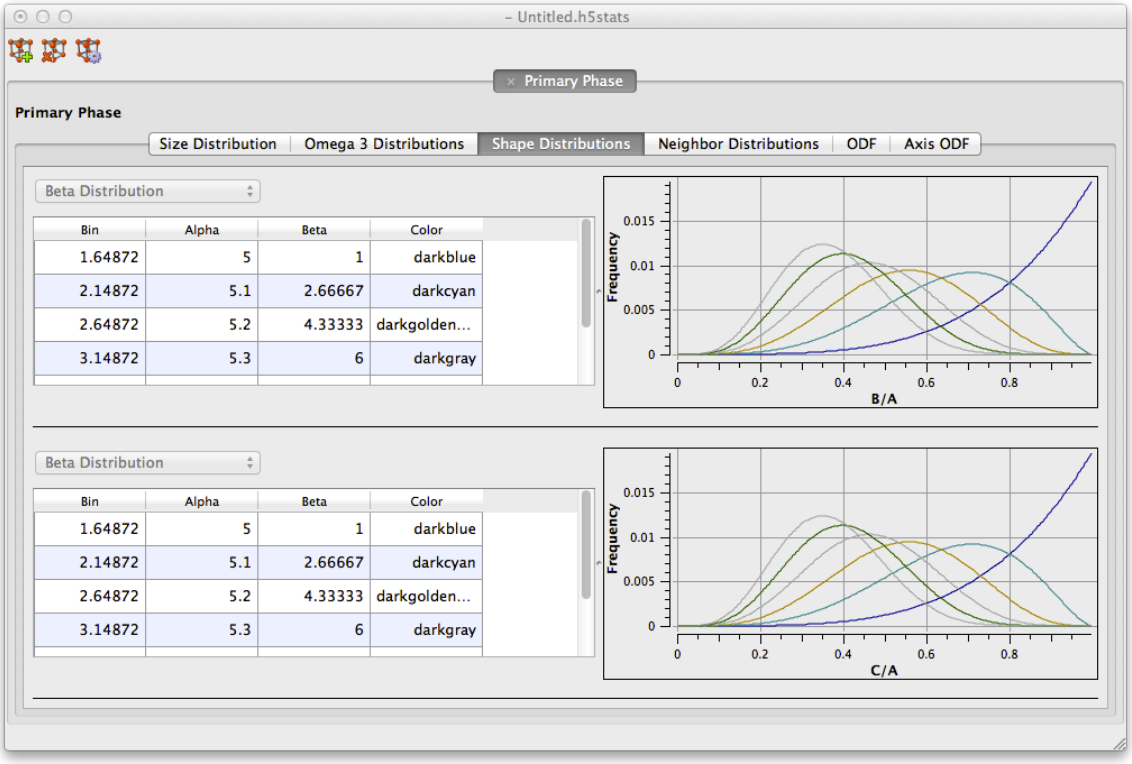
- Bin: this column is calculated from the size distribution and cannot be changed.

- Alpha: this is the alpha parameter of a Beta distribution. Omega 3 is normalized and can only be between 0 and 1, so the Beta distribution is a good fit

- Beta: this is the beta parameter of the Beta distribution.

- Color: this allows the user to change colors of the curves for image creation or easier identification during stats generation

Shape Distribution Tab



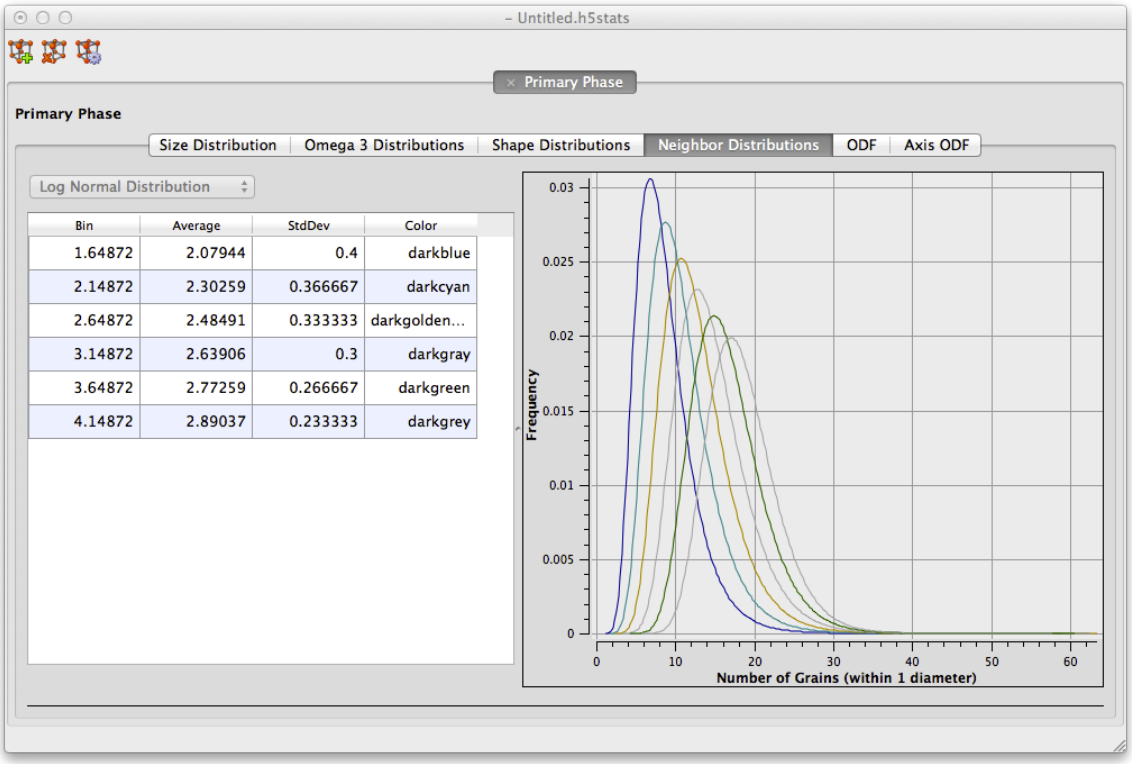
- Bin: this column is calculated from the size distribution and cannot be changed.

- Alpha: this is the alpha parameter of a Beta distribution. B/A, C/A and C/B are normalized and can only be between 0 and 1, so the Beta distribution is a good fit

- Beta: this is the beta parameter of the Beta distribution.

- Color: this allows the user to change colors of the curves for image creation or easier identification during stats generation

Neighbor Distribution Tab



- Bin: this column is calculated from the size distribution and cannot be changed.

- Alpha: this is the alpha parameter of a Power Law distribution.

  - this is the exponent of a Power Law Distribution

- Beta: this is the beta parameter of a Power Law distribution.

- Color: this allows the user to change colors of the curves for image creation or easier identification during stats generation

ODF Tab

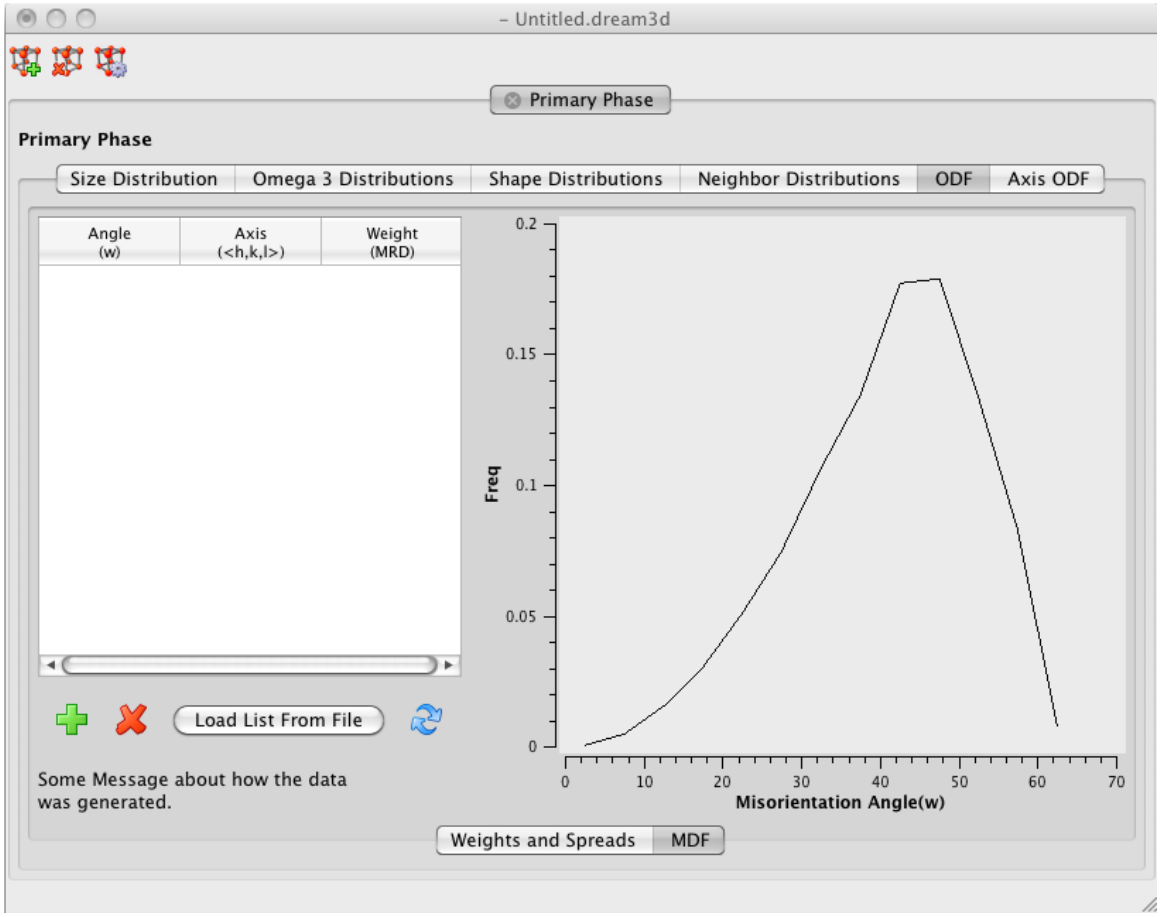
Weights and Spreads Sub-Tab

- Euler 1-3: these are the Euler angles that define an orientation that the user would like to increase in weight.
- Weight: this is the weight in MRD (multiples o
- Sigma: this is the spread to use in blurring out the orientation chosen. The value corresponds to the number of bins in Rodrigues (orientation) space it takes for the MRD value entered in the Weight column to reduce to 0.0 (decreasing quadratically from the bin of the entered orientation).
- Calculate ODF Đ this builds the ODF and then creates pole figures (PFs) for the user to inspect.

Pole Figure (PF) Sub-Tabs

- There are three PFs formed for each of the crystal structures that can be chosen (though they are of different directions for the different crystal structures).

MDF Sub Tab This sub-tab will display the baseline MDF for the generated ODF. The implemented algorithm proceeds by randomly sampling pairs of orientations from the ODF and calculating the misorientation (axis-angle only). Only the angle is plotted in the misorientation distribution plot. The user can also add axis-angle pairs to increase in weight.

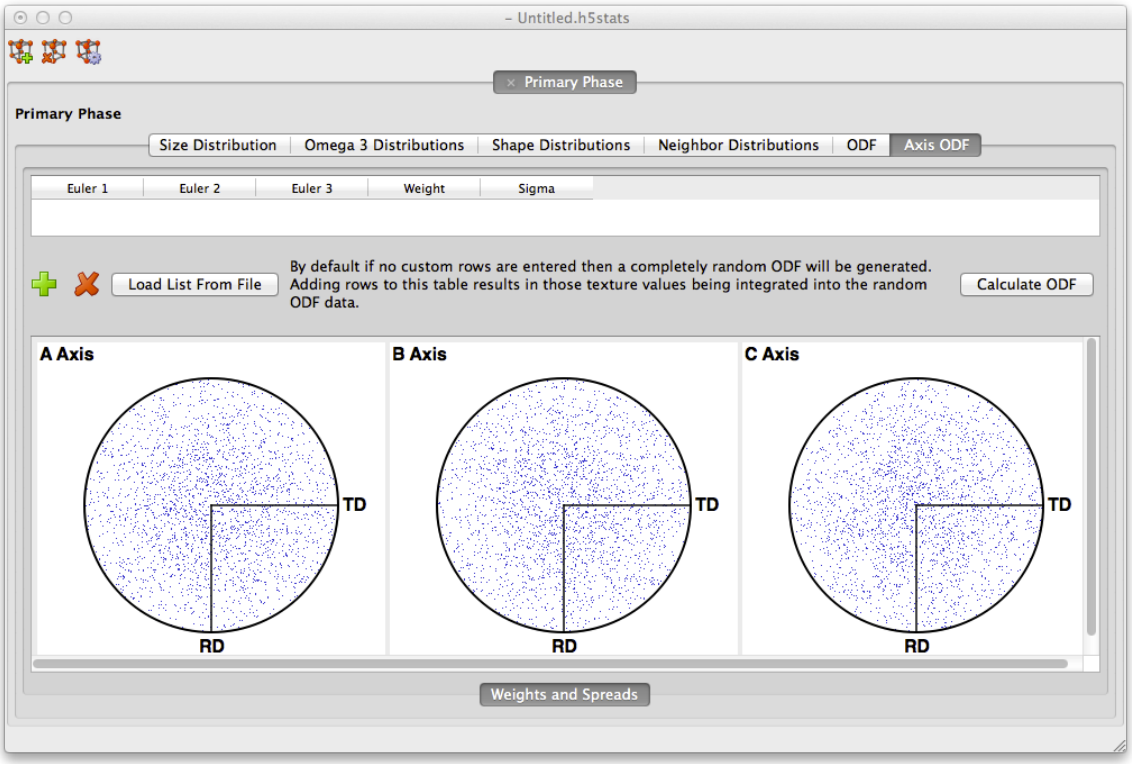


- Angle Ð this is the angle of the misorientation to increase in weight.

- Axis Ð this is the axis of the misorientation to increase in weight. If the crystal structure being used for the phase is Hexagonal, then this axis is in the 3-index, orthogonal convention, not the true (hkil) convention.

- Weight Ð this is the weight in units of MRD (multiples of random) of the entered misorientation.

Axis ODF Tab



Weights and Spreads Sub-Tab

- Euler 1-3 Ð these are the Euler angles that define an orientation that the user would like to increase in weight.
- Weight Ð this is the weight in MRD (multiples of random) to be assigned to the orientation listed
- Sigma Ð this is the spread to use in blurring out the orientation chosen. The value corresponds to the number of bins in Rodrigues (orientation) space it takes for the MRD value entered in the Weight column to reduce to 0.0 (decreasing quadratically from the bin of the entered orientation).
- Calculate ODF Ð this builds the ODF and then creates pole figures (PFs) for the user to inspect.

Pole Figure (PF) Sub-Tabs

There are three pole figures formed, which correspond to the location of the 3 principal axes of the grains to be generated (i.e a $>$ b $>$ c).

CHAPTER 10

Created Array Index

Active (FieldData)

- Minimum Number of Neighbors Filter (Processing->Cleanup)
- Minimum Size Filter (All Phases) (Processing->Cleanup)
- Minimum Size Filter (Per Phase) (Processing->Cleanup)
- Segment Fields (Misorientation) (Reconstruction->Segmentation)
- Segment Fields (Scalar) (Reconstruction->Segmentation)
- Segment Fields (C-Axis Misorientation) (Reconstruction->Segmentation)
- Merge Twins (Reconstruction->Grouping)
- Reconstruct Prior Beta Grains (Reconstruction->Grouping)
- Identify MicroTexture (C-Axis Misorientation) (Reconstruction->Grouping)
- Crop Volume (Sampling->Croping/Cutting)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Insert Precipitate Phases (SyntheticBuilding->Packing)
- Establish Matrix Phase (SyntheticBuilding->Packing)

AspectRatios (FieldData)

- Find Field Shapes (Statistics->Morphological)

AvgCAxes (FieldData)

- Find Average C-Axis Orientations (Statistics->Crystallographic)

AvgQuats (FieldData)

- Read YS Choi Abaqus Vtk Output File (IO->Input)
- Find Field Average Orientations (Statistics->Crystallographic)
- Match Crystallography (SyntheticBuilding->Crystallography)
- Jumble Orientations (SyntheticBuilding->Crystallography)

AxisEulerAngles (FieldData)

- Find Field Shapes (Statistics->Morphological)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Insert Precipitate Phases (SyntheticBuilding->Packing)

AxisLengths (FieldData)

- Find Field Shapes (Statistics->Morphological)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Insert Precipitate Phases (SyntheticBuilding->Packing)

BiasedFields (FieldData)

- Find Biased Fields (Bounding Box) (Generic->Spatial)

Centroids (FieldData)

- Find Field Centroids (Generic->Misc)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Insert Precipitate Phases (SyntheticBuilding->Packing)

CrystalStructures (EnsembleData)

- Read Ensemble Info File (IO->Input)
- Read Legacy DREAM.3D H5Voxel File (IO->Input)
- Read YS Choi Abaqus Vtk Output File (IO->Input)

EquivalentDiameters (FieldData)

- Find Field Sizes (Statistics->Morphological)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Insert Precipitate Phases (SyntheticBuilding->Packing)

EulerAngles (CellData)

- Read Field Info File (IO->Input)
- Read Legacy DREAM.3D H5Voxel File (IO->Input)
- Read YS Choi Abaqus Vtk Output File (IO->Input)
- Match Crystallography (SyntheticBuilding->Crystallography)
- Jumble Orientations (SyntheticBuilding->Crystallography)

EulerAngles (FieldData)

- Read Field Info File (IO->Input)
- Find Field Average Orientations (Statistics->Crystallographic)
- Match Crystallography (SyntheticBuilding->Crystallography)
- Jumble Orientations (SyntheticBuilding->Crystallography)

EulerColor (CellData)

- Generate Euler Colors (Generic->Misc)

FlatImageData (CellData)

- Flatten Image (Processing->Image)

GBEuclideanDistances (CellData)

- Find Euclidean Distance Map (Statistics->Morphological)

GoodFields (FieldData)

- Single Threshold (Field Data) (Processing->Threshold)
- Multi Threshold (Field Data) (Processing->Threshold)

GoodVoxels (CellData)

- Single Threshold (Cell Data) (Processing->Threshold)
- Multi Threshold (Cell Data) (Processing->Threshold)
- Segment Fields (Misorientation) (Reconstruction->Segmentation)
- Segment Fields (C-Axis Misorientation) (Reconstruction->Segmentation)

GrainAvgCAxisMisorientations (FieldData)

- Find Field Reference C-Axis Misorientations (Statistics->Crystallographic)

GrainAvgMisorientations (FieldData)

- Find Field Reference Misorientations (Statistics->Crystallographic)

GrainIds (CellData)

- Read Dx File (Grain Ids) (IO->Input)
- Read Legacy DREAM.3D H5Voxel File (IO->Input)
- Read Ph File (Grain Ids) (IO->Input)
- Read YS Choi Abaqus Vtk Output File (IO->Input)

- Read Vtk File (STRUCTURED_POINTS) Grain Ids Only (IO->Input)
- Segment Fields (Misorientation) (Reconstruction->Segmentation)
- Segment Fields (Scalar) (Reconstruction->Segmentation)
- Segment Fields (C-Axis Misorientation) (Reconstruction->Segmentation)

GrainReferenceCAxisMisorientations (CellData)

- Find Field Reference C-Axis Misorientations (Statistics->Crystallographic)

GrainReferenceMisorientations (CellData)

- Find Field Reference Misorientations (Statistics->Crystallographic)

IPFColor (CellData)

- Generate IPF Colors (Generic->Crystallography)

KernelAverageMisorientations (CellData)

- Find Kernel Average Misorientations (Statistics->Crystallographic)

NearestNeighbors (CellData)

- Find Euclidean Distance Map (Statistics->Morphological)

NeighborList (Field)

- Find Field Neighbors (Statistics->Morphological)

Neighborhoods (FieldData)

- Find Field Neighborhoods (Statistics->Morphological)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Insert Precipitate Phases (SyntheticBuilding->Packing)

NumCells (FieldData)

- Find Field Sizes (Statistics->Morphological)
- Insert Precipitate Phases (SyntheticBuilding->Packing)

NumFields (EnsembleData)

- Find Number of Fields (Statistics->Morphological)
- Insert Precipitate Phases (SyntheticBuilding->Packing)

NumNeighbors (FieldData)

- Find Field Neighbors (Statistics->Morphological)

Omega3s (FieldData)

- Find Field Shapes (Statistics->Morphological)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Insert Precipitate Phases (SyntheticBuilding->Packing)

ParentIds (CellData)

- Merge Twins (Reconstruction->Grouping)
- Reconstruct Prior Beta Grains (Reconstruction->Grouping)
- Identify MicroTexture (C-Axis Misorientation) (Reconstruction->Grouping)

ParentIds (FieldData)

- Merge Twins (Reconstruction->Grouping)
- Reconstruct Prior Beta Grains (Reconstruction->Grouping)
- Identify MicroTexture (C-Axis Misorientation) (Reconstruction->Grouping)

PhaseTypes (EnsembleData)

- Read Ensemble Info File (IO->Input)
- Read Legacy DREAM.3D H5Voxel File (IO->Input)

Phases (CellData)

- Read Field Info File (IO->Input)
- Read Legacy DREAM.3D H5Voxel File (IO->Input)
- Read YS Choi Abaqus Vtk Output File (IO->Input)

Phases (FieldData)

- Find Field Phases (Generic->Misc)
- Read Field Info File (IO->Input)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Insert Precipitate Phases (SyntheticBuilding->Packing)
- Establish Matrix Phase (SyntheticBuilding->Packing)

Poles (FieldData)

- Find Schmid Factors (Statistics->Crystallographic)

QPEuclideanDistances (CellData)

- Find Euclidean Distance Map (Statistics->Morphological)

Quats (CellData)

- Find Cell Quaternions (Generic->Crystallography)
- Read YS Choi Abaqus Vtk Output File (IO->Input)

Schmids (FieldData)

- Find Schmid Factors (Statistics->Crystallographic)

SharedSurfaceAreaList (Field)

- Find Field Neighbors (Statistics->Morphological)

SlipSystems (FieldData)

- Find Schmid Factors (Statistics->Crystallographic)

SurfaceFields (FieldData)

- Find Surface Fields (Generic->Spatial)
- Read YS Choi Abaqus Vtk Output File (IO->Input)
- Find Field Neighbors (Statistics->Morphological)

SurfaceMeshNodeType (VertexData)

- M3C Surface Meshing (Slice at a time) (SurfaceMeshing->Generation)

SurfaceVoxels (CellData)

- Find Boundary Cells (Generic->Spatial)
- Find Field Neighbors (Statistics->Morphological)

TJEuclideanDistances (CellData)

- Find Euclidean Distance Map (Statistics->Morphological)

Volumes (FieldData)

- Find Field Shapes (Statistics->Morphological)
- Find Field Sizes (Statistics->Morphological)
- Pack Primary Phases (SyntheticBuilding->Packing)
- Match Crystallography (SyntheticBuilding->Crystallography)
- Insert Precipitate Phases (SyntheticBuilding->Packing)